FIRST ORDER EXPRESSIBILITY AS A NEW COMPLEXITY MEASURE

Neil Immerman

Ph.D. Thesis

TR 80-432

Department of Computer Science
Cornell University
Ithaca, N.Y. 14853

FIRST ORDER EXPRESSIBILITY AS A NEW COMPLEXITY MEASURE

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment for the Degree of

Doctor of Philosophy

by

Neil Immerman

August, 1980

Biographical Sketch

Neil Immerman was born on November 24, 1953 in Manhasset, New York. He grew up on Long Island. In the summer of 1969 Neil attended a National Science Foundation sponsored summer program at the University of New Hampshire at which he met some respectable people who enjoyed their days doing mathematics.

He visited and later taught at Hampshire College Summer Studies in Mathematics, a descendant of the UNH program. This idealistic six week session in which math is taught and created in an entirely cooperative and supportive atmosphere was a major influence on Neil. It was also the place at which he met Susan Landau to whom he was married sometime later.

In 1974 Neil graduated Yale University *summa cum laude* with both B.S. and M.S. degrees in mathematics. The M.S. emphaisized mathematical logic.

Needing a break from school, Neil spent the next year in Boston programming and designing mini-computer controlled telephone switching systems for GTE-Sylvania. During that year he learned English Change Ringing from Geoffrey Davies at the Church of the Advent in Beacon Hill.

In 1975, after winning a National Science Foundation graduate fellowship, Neil enrolled in Cornell University. Cornell which was founded in 1865 is nestled in the scenic beauty of the finger lakes region of upstate New York. Neil began Cornell planning to do a thesis in Mathematical logic, but, five years later, has written his dissertation in Computer Science.

Acknowledgements

Ever so much thanks to Juris Hartmanis for his kind advice, concern and suggestions.

Thanks of course to Anil Nerode my chairman for watching out for me while at the same time giving free rein.

Thanks to John Hopcroft, Albert Meyer, and Michael Morley for invaluable and enjoyable technical discussions.

Thanks to Larry Carter for showing me Turing machines long ago and also for sending me Ron's paper.

It is hard to list all the fellow graduate students and other friends and relations who have been supportive technically and otherwise, especially during those dark middle years when I thought I would never prove anything and would have to be a graduate student forever, but you know who you are. Thanks.

Thanks to MIT who let me visit and type this thesis over the summer of 1980.

Finally thanks to the NSF for the financial support provided by a graduate fellowship and later by grant number MCS 78-00418.

TABLE OF CONTENTS

Introduction

The main purpose of Computational Complexity is to answer the question, "How difficult is it to perform a given task on a computer?" An algorithm to perform the task provides an upper bound on its complexity. However to prove a lower bound one must show that *no conceivable algorithm* for the task can run in, say, n^{17} steps.

In the hope of proving lower bounds we introduce a new complexity measure. Most measures count how much of some computational resource (e.g. time or memory space) is needed to *check* whether an input has a certain property, C. Instead, we examine the size of sentences and number of variables needed to *express* C in first order predicate logic.

As we will see, the minimum size of sentences expressing C is closely related to the space complexity of C. Also the number of variables needed is tied to the logarithm of the time complexity. Thus the difficulty of expressing conditions in the language of mathematics is a good measure of their computational complexity. We will exploit Ehrenfeucht-Fraisse games to prove lower bounds on the number of quantifiers and variables needed. These lower bound techniques are a new addition to the more usual methods of complexity theory involving complete sets or diagonalization. We believe that it makes more intuitive sense to prove a lower bound (by induction, say) on the number of quantifiers needed to express a certain property, than on the number of Turing machine tape cells needed to check if the property holds for a given input.

The original stimulus for this dissertation came from work by Fagin (see [Fag74]). He proved the following:

Theorem (Fagin): A set, S, of structures is in NP if and only if there exists a sentence, F, with the following properties:

1. $F = \exists P_1 \ldots \exists P_k \Phi(P_1 \ldots P_k)$, where $P_1 \ldots P_k$ are predicate symbols and Φ is a first order sentence.

2. Any structure, G is in S iff G satisfies F.

Thus a property is in NP just if it is expressible by a second order existential sentence. It is difficult to show lower bounds for the expressibility of second order sentences. Instead we examine first order sentences which, we found, mimic computations much more closely. Considering, for example, graph problems, the length of the shortest sentence which says, "G is connected," grows as the logarithm of the size of G. It is not a coincidence that this is also the space needed by a Turing machine to test if G is connected. To study this growth of sentences we introduce the complexity measure Size which will be defined in Chapter 1. Informally, a set, S, of structures is in Size[f(n)] if membership in S for those structures of size less than or equal to n can be expressed by a sentence with f(n) symbols.

These sentences are written in the language of the given structures. For example if we are dealing with graph problems then the quantifiers range over the vertices and there is a single relation symbol, E(-,-), representing the edge relation. This language seems sufficient to describe "natural" problems on graphs, but to simulate an arbitrary Turing Machine computation we must give the language access to an ordering of the universe. We let Size[f(n)] be the family of properties expressible with f(n) quantifiers in a language that includes Suc(-,-), a successor relation. We can now show that NSPACE[f(n)] is contained in Size[$(f(n))^2$/logn].

We will say that C is in Size[f(n)] only if the sequence of sentences expressing C is easy to generate. This added assumption of uniformity (in the sense of Borodin, see [Bor77]) allows us to prove Size[g(n)] $\subseteq$ DSPACE[g(n)logn], and thus:

$$\text{NSPACE}[f(n)] \subseteq \text{Size}[(f(n))^2/\log n] \subseteq \text{DSPACE}[(f(n))^2] .$$

Note that our lower bounds will not consider the uniformity; they may be interpreted in the strongest possible sense. When we show that C is not in Size[f(n)] we actually prove that *no sentence with f(n) symbols* expresses C for structures of size n.

In chapter 2 we consider restricting the number of distinct variables used in our sentences. We define Var &Sz[v(n),z(n)] to be the class of properties uniformly expressible by sentences with exactly v(n) variables, and O[z(n)] symbols. Var &Sz corresponds very closely to Alternating Space &Time. In particular we show that:

$$\text{ASPACE \&TIME}[s(n),t(n)] \subseteq \text{Var \&Sz}[k^{s(n)}/\log(n),t(n)] \subseteq \text{ASPACE \&TIME}[s(n),t(n)\log(n)]$$

Once the idea of counting distinct variables was raised it was natural to relax the size restriction. Define Var[k] to be those properties expressible with a constant number of variables. It turns out that Var[k] is identical to polynomial time! The identity between P and Var[k] is a very pleasing result both because it indicates that first order expressibility is a fruitful view of complexity, and because it is another demonstration of the fundamental importance and model independent nature of P.

The quantifier rank of a sentence T is the depth of nesting of quantifiers in T; thus a sentence with n quantifiers has at most quantifier rank n. In Chapter 3 we consider a two person game with which we prove lower bounds for quantifier rank. An Ehrenfeucht-Fraisse game is played on a pair of structures G,H of the same type. Player I chooses points to show that G and H are different, while Player II matches these points, trying to keep the structures looking the same. A theorem due to Fraisse and Ehrenfeucht says that Player II has a winning strategy for the n move game if and only if G and H agree on all sentences of quantifier rank n. The original treatment of these games appears in [Ehr61] and [Fra54].

Ehrenfeucht-Fraisse games provide a lower bound technique for Size as follows: Given some property, C, we find structures G and H of size n such that G satisfies C but H does not. We then show that Player II has a winning strategy for the f(n) move game on G and H. It follows that G and H agree on all sentences of quantifier rank f(n) and thus in particular no sentence with f(n) symbols can express the property C. Thus we have shown that C is not in Size[f(n)]. These combinatorial games provide very sharp lower bounds. We show for example that while quantifier rank log n suffices to express the graph property, "There is a path from point a to point b," quantifier rank log(n)-2 is insufficient!

In Section 3.3 we present a more sophisticated game argument. We show that without successor

quantifier rank $\log^k(n)$ is insufficient to describe a set recognizable in polynomial time. If our proof went through for the language with successor we would have shown that PTIME is not contained in $U_{k=1,2,\ldots}$ SPACE[$\log^k(n)$].

In Chapter 4 we modify the Ehrenfeucht-Fraisse games to what we call Alternating Pebbling games. We show that Player II wins the k pebble, m move game on G and H if and only if G and H agree on all sentences with k distinct variables and quantifier rank m. We then play the game, proving lower bounds on what can be said in Var(w.o. Suc)[k], i.e. with k variables, but without successor. We show for example that Clique(k), the set of graphs having a complete subgraph of size k, is not in Var(w.o.Suc)[k-1]. Obviously, however, Clique(k) is in Var(w.o. Suc)[k]. This may be thought of as an intuitive argument that testing for a k clique requires looking at all k-tuples of vertices and thus requires DTIME[n^k].

Making the above results go through with successor is a major open problem. We show that quantifier rank is no longer the right thing to check. In a language with successor, any property whatsoever of graphs of size n can be expressed by a sentence with 2^{n^2} quantifiers but quantifier rank only log n. To make matters worse two ordered graphs G and H satisfy all of the same 3 variable, $3\log(n)$ quantifier sentences in the language with successor only if they are identical. This is as expected because G and H are indistinguishable to all log space Turing machines only if they are identical. The proof is the same in both cases: the machine or the short sentence can check if vertex 3 is connected to vertex 17. G and H agree on all such tests only if they are identical.

We conclude by proposing a few possible techniques for adding successor to the above result and thus proving that USize[$\log^k(n)$]$\neq$P. The most promising one at present is a modification of Ehrenfeucht-Fraisse games such that Player I wins the k move game if and only if a given property is expressible with k quantifiers and Suc(-,-). This new game is combinatorially much more complex than the Ehrenfeucht-Fraisse game and so we are by no means proficient at playing it. And yet we wanted to present, as a point of departure for future research, what may become a viable technique for proving lower bounds.

Chapter 1

Definitions and Motivations

We propose to study the complexity of a condition, C, by asking, "How difficult is it to express C?" For this expression we choose the natural first order language of the objects under consideration.

This chapter is organized as follows: In Section 1 we give the necessary logical definitions. Then in Section 2 we motivate the definition for Size[f(n)]. Finally, in Section 3 we prove that the size of a first order sentence needed to express a property, C, is polynomially related to the space needed to check if C holds for a given input.

Section 1.1: Review of some notions from logic.

A structure, $S = \langle U, c_1^S \ldots c_k^S, P_1^S \ldots P_n^S \rangle$, consists of a universe, U, certain constants, $c_1^S \ldots c_k^S$, from U, and certain relations, $P_1^S \ldots P_n^S$, on U.

A similarity type, $\tau = \langle c_1 \ldots c_k, P_1 \ldots P_n \rangle$, is a sequence of constant *symbols* and relation *symbols*.

As an example let G be a directed graph with two specified points s and d. Thus $G = \langle V, E^G, s^G, d^G \rangle$ is a structure of type $\tau_g = \langle E, s, d \rangle$, where V is the set of vertices of G, and E^G is G's edge relation.

If τ is any type then $L[\tau]$, the language of τ, is the set of all sentences built up from the symbols of τ using $\wedge$, or, $\neg$, $\Rightarrow$, variables x,y,... , and the quantifiers $\exists$ and $\forall$.

A sentence, F, in $L[\tau]$ is given meaning by a structure, S, of type τ as follows: The symbols from τ are interpreted by the constants and relations in S. The quantifiers in F range over the elements of the universe of S.

For example, let $A = \forall x\,[x = d \text{ or } \exists y E(x,y)]$. A is in $L[\tau_g]$. Furthermore, G satisfies A (in symbols, $G \models A$) iff each vertex of G except d^G has an edge coming out of it. Henceforth we will omit the superscript G for the sake of readability.

The quantifier rank of sentence F, (qr[F]), is the depth of nesting of quantifiers in F. Inductively:

$$qr[(\forall x)B] = qr[(\exists x)B] = qr[B]+1$$
$$qr[B \,\&\, C] = qr[B \text{ or } C] = \max(qr[B], qr[C]).$$

For example, for $A = \forall x[\,(\exists y\, P(x,y)) \,\&\, \forall z \forall w(Q(x,z) \text{ or } L(z,w))]$, $qr[A] = 3$.

The number of elements in the universe of S is abbreviated $|S|$. For graphs $|G|$ is the number of vertices of G.

We use fairly standard notions from complexity, see e.g. [AHU74]. Define DTIME[f(n)] to be the family of problems accepted by a Turing machine in worst case deterministic time f(n) for inputs of size n. Similarly define NTIME, DSPACE, NSPACE; nondeterministic time, deterministic space, and nondeterministic space respectively.

Throughout this paper a problem will be some subset of all structures of a particular type, τ. Some examples of graph problems are Connectivity -- the set of all connected graphs, Planarity -- the set of all planar graphs, etc.

We will assume that any given structure has universe 1,2 ... n. Thus an input consists of a table of all relations and constants placed on a read-only input tape. For graphs, for example, the input is the adjacency matrix, a binary string of length n^2. For simplicity we will define the size of a structure to be n, the size of its universe. Note that we are not restricting ourselves by only considering problems which are sets of structures: any binary string may be thought of as a structure with one monadic relation.

Section 1.2: Motivations and Definition of Size[f(n)].

To motivate the definitions for variable and size expresibility we now consider a stepwise refinement of sentences expressing a specific problem. Let GAP be the set of directed graphs G with specified points s and d such that there is a path in G from s to d. In symbols,

$$GAP = \{ G \mid s \rightarrow^* \rightarrow d \}$$

GAP is known to be complete for NSPACE[logn] (see [Sav73]). We show in Chapter 5 that GAP is complete in a very strong sense -- every problem C in NSPACE[logn] has a first order sentence translating all instances of C into instances of GAP.

To express GAP we will write down formulas $P_n(x,y)$, meaning, "There is a path of length at most n from x to y." We define P_n by induction as follows:

$$P_1(x,y) = (x=y) \text{ or } E(x,y) \tag{1.1}$$

$$P_{2k}(x,y) = \exists z [P_k(x,z) \& P_k(z,y)] \tag{1.2}$$

Equation (1.2) defines P_{2k} in a way that increases the quantifier depth by one each time k is doubled. However P_k is writen twice on the right so the size of this P_{2k} is twice the size of P_k. We can alleviate this problem using the "abbreviation trick" (see e.g. [FiRa74]). The trick uses universal quantifiers to permit us to write P_k only once on the right. Thus,

$$P_{2k}(x,y) = \exists z \forall u \forall v [u=x \& v=z \text{ or } u=z \& v=y] \Rightarrow P_k(u,v) \tag{1.3}$$

We have now written P_n in size O[logn], thus proving that GAP is in Size[logn], to be defined. Let's start with a tentative

Definition: A set C of structures of type τ is expressible (w.o. successor) in size $s(n)$, (in symbols, C is in Size(w.o. Suc)[$s(n)$]), if there exists a constant k, and a uniform sequence of sentences $F_1, F_2, \ldots$ from $L[\tau]$ such that:

a. For all structures, G, of type τ, if $|G| \leq n$, then:

$$G \text{ is in } C \iff G \models F_n .$$

b. F_n has fewer than $k \cdot z(n)$ symbols.

As Ruzzo has shown in [Ru79b] uniformity conditions may be greatly varied without significantly changing a definition. The following conditon will suffice in what follows:

Uniformity Condition (*): The map $n:\rightarrow F_n$ is generable in DTIME[z(n)].

In the above discussion we have proved:

Theorem 1.1: GAP is in Size(w.o. Suc)[logn].

Although the complete problem GAP is in Size(w.o. Suc)[logn] it is not true that NSPACE[logn] is contained in this class. As we will see in Chapter 3, this fails in a rather spectacular way: the regular set. EVEN = { G | G has an even number of vertices }, is not in Size(w.o. Suc)[logn].

To allow them to simulate Turing machines it suffices to give the sentences access to the numbering of the vertices which the machines already have. Thus we will consider properties expressible with an arbitrary successor relation, Suc(-,-). Suc(x,y) means that y comes just after x in the numbering of the elements of the universe.

A similar Suc relation is discussed in [Sav73]. Savitch shows that his pebble automata cannot accept GAP without Suc. However Theorem 1.1 suggests that our sentences do not need Suc to express "natural" graph problems.

Definition: A set C of structures of type τ is expressible in size z(n), (in symbols, C is in Size[z(n)]), if there exists a constant k, and a uniform sequence of sentences $F_1, F_2, \ldots$ from $L[\tau \cup \{Suc\}]$ such that:

a. For all structures, G, of type τ, if $|G| \leq n$, and for all $Suc_0(\cdot,\cdot)$ a valid successor relation on the universe of G,

$$G \text{ is in } C \iff \langle G, Suc_0 \rangle \models F_n .$$

b. F_n has fewer than $k \cdot z(n)$ symbols.

Thus a property, C, is in Size[z(n)] if there is a uniform sequence of sentences of size z(n) from $L[\tau \cup \{Suc\}]$ which give the same answer for any successor relation and express C. We feel that this addition of Suc is not at all natural but it captures the notion of what properties a Turing machine can check. It is easy to list the Turing machines which check properties of ordered graphs, but we see no obvious way to list those machines which happen to check graph properties independent of the given ordering.

Section 1.3: Size is closely related to SPACE.

Now that we have added Suc to our language it follows that NSPACE[logn] is contained in Size[logn], and indeed:

Theorem 1.2: Let $s(n) \geq \log(n)$. Then:

$$NSPACE[s(n)] \subseteq Size[s(n)^2/\log(n)] \subseteq DSPACE[s(n)^2]$$

proof: We start with the second inclusion. We show that Size[g(n)] is contained in DSPACE[g(n)log(n)]. Given input structure G of size n we can certainly generate F_n in the given space by the uniformity condition. Check the truth of a g(n) quantifier sentence in DSPACE[g(n)log(n)] as follows: Cycle through the sentence with all possible values of the quantified variables. If F_n is of the form $\exists x P(x)$ then we test the truth of P(1), ... ,P(n). Each variable requires log n bits and at most g(n) of them must be remembered at once. When all the variables in F_n have been replaced by constants its truth may be checked as we generate it with no additional space required.

The first inclusion: We will code a Turing machine instantaneous description (ID) of size s(n) with O(s(n)/log(n)] variables. The idea is that each variable takes on a value from 1 to n and so may be thought of as log(n) bits. Details of this coding are given in Lemma 1.2a. An ID consists of a state, the location of the input head, and the s(n)/log(n) variable work tape. The read head requires a constant number of variables -- for a graph two variables u,v give that element of the adjacency matrix being

scanned. It is 1 or 0 according as E(u,v) is true or false. One use of Suc is to efficiently code log(n) bits of worktape into one variable, but the essential use is to say that the read head has moved one space to the right.

We can now write $P_1(ID_a, ID_b)$ meaning that ID_b follows from ID_a in one step of M. This is just a constant size disjunction over the possible states, input symbols and work tape symbols. We will see in Lemma 2a that the symbol being read by the work tape can be determined with a size log(n) formula. The rest of P_1 requires only a constant number of symbols.

As in the proof that GAP is in Size[logn] we can now assert that there is a computation path of length $c^{s(n)}$ using O[s(n)] ID's. Thus the total size of P_n is $s(n)^2/\log(n)$ as required. ∎

To finish the proof of Theorem 2 we must show that O[logn] symbols suffice to calculate with vertex numbers. We code an s(n) bit work tape with s(n)/log(n) pairs of vertices $\langle x_j, y_j \rangle$. All of the y_j's but one will be 0. The nonzero y_j will be vertex number 2^i when the work head is looking at bit i of x_j.

Let On(x,y) mean that $y = 2^i$ and bit i of x is on. We will see in Lemma 2a that On(x,y) may be written with O[logn] quantifiers. It follows that $P_1(\cdot,\cdot)$ may also; just say that some y_j is nonzero, the nonneighboring $\langle x,y \rangle$ pairs are unchanged, and $\langle x_j, y_j \rangle$ and its neighbor (in case the work head happens to move to an adjacent block) are changed as per the rules of M.

Lemma 1.2a: On(x,y) may be written in Size[logn].

proof: We build up to "On" with a sequence of inductive definitions, repeatedly using the abbreviation trick as in [FiRa74]. Thus each of the following formulas may be written in Size[logn] by using the previous one.

(a) $\quad Plus_n(x,y,z) = (x \leq n) \,\&\, (x + y = z)$

(b) $\quad Q_n(x_1 \ldots x_{\log n}) = (x_1 = 1) \,\&\, \wedge_{0<i<\log n} (x_{i+1} = x_i + x_i)$

(c) $\quad R_n(y_1 \ldots y_{\log n}) = \exists x_1 \ldots x_{\log n} \left(Q(x_1 \ldots x_{\log n}) \;\&\; \wedge_{0<i<\log n+1} (y_i = 0 \text{ or } y_i = x_i) \right)$

(d) $\quad S_n(z, y_1 \ldots y_{\log n}) = R_n(y_1 \ldots y_{\log n}) \,\&\, z = y_1 + \ldots + y_{\log n}$

(e) $On(z,x) \quad = \exists y_1 \ldots y_{\log n} \left(S_n(z, y_1 \ldots y_{\log n}) \;\&\; \bigvee_{0 \le i \le \log n + 1} (x = y_i \neq 0) \right)$

For example, Plus is defined as follows:

$$Plus_1(x,y,z) \quad = \quad [x = 0 \;\&\; y = z] \quad \text{or} \quad [x = 1 \;\&\; Suc(y,z)]$$

$$Plus_{2k}(x,y,z) \quad = \quad \exists u\, v\, w \left(Plus_k(u,v,x) \;\&\; Plus_k(u,y,w) \;\&\; Plus_k(v,w,z) \right)$$

As before we can use the abbreviation trick to write $Plus_k$ only once on the right. Note that we use the symbols 0 and 1 for convenience but they are of course definable from Suc.

Note also that to get each next equation one cannot simply use the previously defined formula. For example to get Q_n a first try might be:

$$Q_n(x_1 \ldots x_{\log n}) \quad = \quad (x_1 = 1) \;\&\; Plus_n(x_1, x_1, x_2) \;\&\; \ldots \;\&\; Plus_n(x_{\log n - 1}, x_{\log n - 1}, x_{\log n})$$

However this requires $\log^2(n)$ quantifiers. We must conglomerate $Plus_n$ and Q_n into

$$C_n(x_1 \ldots x_{\log n}, u, v, w) = \quad Q_n(x_1 \ldots x_{\log n}) \;\&\; Plus_n(u,v,w)$$

and define them simultaneously, using the abbreviation trick. ∎

Chapter 2

Variables & Size versus Time & Space

Reading some papers by Ruzzo ([Ruz79a], [Ruz79b]), on simultaneous resource bounds, we tried to find analagous results for first order expressibility. First we reexamined our proof of Theorem 1.2 for $s(n) = \log(n)$, i.e:

$$\text{NSPACE}[\log n] \subseteq \text{Size}[\log n] \subseteq \text{DSPACE}[\log^2(n)],$$

and noticed that only a constant number of variables were needed. Furthermore while the existential quantifiers range over the elements of the universe of the input, (i.e. 1 to n), the universal quantifiers could be boolean. Thus we let $\underline{\text{Var}\ \&S_\ell[v(n),z(n)]}$ be the class of properties uniformly expressible with exactly $v(n)$ variables and size $O[z(n)]$. Also let $\underline{\text{BUVar}\ \&S_\ell[v(n),z(n)]}$ be the same class with the additional restriction that the universal quantifiers are boolean. We show that:

$$\text{NSPACE}[\log n] \subseteq \text{BUVar}\ \&S_\ell[k,\log n] \subseteq \text{Var}\ \&S_\ell[k,\log n] \subseteq \text{DSPACE}[\log^2(n)].$$

Savitch's simulation of NSPACE[logn] by DSPACE[$\log^2(n)$] may be optimal, but one way of thinking of the difference between the classes is that DSPACE[$\log^2(n)$] can simulate log(n) universal quantifiers ranging from 1 to n. We conjecture that all three containments in the above chain are proper, but none are known to be. (Note that for readability we are omitting unions over k, for example Var $\&S_\ell[k,\log n]$ abbreviates $\bigcup_{k=1,2,\ldots}$ Var $\&S_\ell[k,\log n]$.)

It turns out that BUVar $\&S_\ell[k,\log n]$ is identical to the natural class Log(CFL) -- those languages log-space reducible to some context free language. We will also see that the third term in the above chain, Var $\&S_\ell[k,\log n]$, is equal to ASPACE $\&\text{Alt}[\log n,\log n]$ -- the class of languages accepted by an ASPACE[logn] Turing machine which makes only O[logn] alternations between existential and universal states.

Once the idea of counting distinct variables was raised it was natural to relax the size restriction. Define $\underline{\text{Var}[k]} = \bigcup_{k=1,2,\ldots} \text{Var}\,\&\text{Sz}[k,n^k]$ -- those properties expressible with a constant number of variables. It turns out that Var[k] is identical to polynomial time!

The identity between P and Var[k] is a very pleasing result both because it indicates that first order expressibility is a fruitful view of complexity, and because it is another demonstration of the fundamental importance and model independent nature of P.

One weakness of the definition of expressibility size in Chapter 1 is that it makes use of the notion of Turing machines in the definition of a "uniform" sequence of sentences. Our feeling at the time was that the uniformity condition was an imperfect attempt to capture the notion that we really had one sentence with a variable number of quantifiers, just as we have the notion of one Turing machine with a variable amount of space. Indeed the use of constantly many variables leads us to the realization that there is a syntactic uniformity -- the n^{th} sentence of a Var $\&\text{S}\mathcal{I}[k,\tau(n)]$ property is just $\tau(n)$ repetitions of a fixed block of k quantifiers. This new definition of uniformity makes Var[k] entirely a notion from logic and thus increases the interest of the fact that it is equal to P.

This chapter is organized as follows: In the first section we continue our refinement of sentences expressing GAP. This leads to a definition of Var $\&\text{S}\mathcal{I}[v(n),\tau(n)]$ -- those properties expressible with $v(n)$ distinct variables in sentences of size $\tau(n)$. In the next section we relate BUVar &Sz to other models of computation that happen to charge more for universal moves than existential ones, namely Auxilliary PDA's, and Ruzzo's Tree-Size Bounded Alternation. In Section 2.3 we generalize these last results to show that alternating space s and time t is identical to expressibility with s boolean variables and size t. From this follows the above mentioned equality, P = Var[k], which we explore in Section 2.4.

Section 2.1: The Var &Sz Measure.

Recall that in Chapter 1 we wrote the size O[logn] sentence, $P_n(x,y)$, which says there is a path of length at most n from x to y:

$$P_n(x,y) = \exists z \forall u \forall w [\, u=x \& v=z \text{ or } u=z \& v=y \,] \Rightarrow P_{n/2}(u,v) \qquad (2.1)$$

Continuing in our refinement notice that when we write $P_{n/2}(u,v)$ we may reuse x,y,z -- their current values are no longer needed. Being slightly wasteful for the sake of clarity, write:

$$P_n(x,y) = \exists z \forall u \forall w \left([u=x \& v=z \text{ or } u=z \& v=y\,] \Rightarrow \exists x \exists y [x=u \& y=v \,\&\, P_{n/2}(x,y)] \right) \qquad (2.2)$$

We have succeeded in expressing GAP by a uniform sequence of sentences, $\{P_n(a,b) \mid n \geq 1\}$, such that P_n has 5 variables and size O[logn]. This suggests the following:

Definition: A set C of structures of type τ is expressible in v(n) variables and size z(n), (in symbols, C is in $Var\&Sz[v,z(n)]$), if there exists a uniform sequence of sentences $F_1, F_2, \ldots$ from $L[\tau \cup \{Suc\}]$ such that:

a. For all structures G of type τ with $|G| \leq n$, and for all $Suc_0(\cdot,\cdot)$ a valid successor relation on the universe of G,

$$G \in C \quad \leftrightarrow \quad \langle G, Suc_0 \rangle \models F_n$$

b. F_n has v(n) distinct variables and a total of O[z(n)] symbols.

As in our uniformity definition for Size, it suffices to require that F_n is easy to generate, i.e.:

Uniformity Condition (*): The map $n \rightarrow F_n$ is generable in DSPACE[v(n)·logn] and DTIME[z(n)].

Of course (*) does not capture our intuitive feeling that the F_n's are all the same sentence with varying numbers of quantifiers. To make the latter notion more precise abbreviate quantifiers with restricted domains as follows:

$$(\exists x \,.\, M)[\,\ldots\,] = \exists x [M \,\&\, \ldots] \qquad \text{(read, "There exists x such that M, ... ,")}$$

$$(\forall x \,.\, M)[\,\ldots\,] = \forall x [M \rightarrow \ldots] \qquad \text{(read, "For all x such that M, ... ,")}$$

Now we can write Equation (2.2) more compactly as:

$$P_n(x,y) = \exists z \forall u (\forall v \,.\, M_3) \exists x (\exists y \,.\, M_5)\, P_{n/2}(x,y) \qquad (2.3)$$

Here $M_3 = [u=x \& v=z \text{ or } u=z \& v=y\,]$, and $M_5 = [x=u \,\&\, y=v]$. Equation (2.3) gives a model

for the following totally syntactical definition of uniformity for Var &Sz[v,z(n)]:

Uniformity Condition (**): There exist constant c and formulas A,B, and quantifier free formulas $M_1 \ldots M_v$ all of which have variables only $x_1 \ldots x_v$ such that:

$$F_n = A\left((Q_1 x_1 . M_1) \ldots (Q_v x_v . M_v) \right)^{c \cdot z(n) \text{ times}} B .$$

We adopt (**) as our definition of uniformity for Var &Sz[v,z(n)] when v is a constant, otherwise we use (*). It follows that GAP is in Var &Sz[5,logn], NSPACE[logn] is in Var &Sz[k,logn], and indeed:

Theorem 2.1: For $s(n) \geq \log(n)$,

$$\text{NSPACE}[s(n)] \subseteq \text{Var \&Sz}[(k \cdot s(n))/\log(n), s(n)^2/\log(n)] \subseteq \text{DSPACE}[s(n)^2]$$

The proof is the same as for Theorem 1.2, using Equation (2.3), rather than (2.1), to express the existence of a computation path.

Let's return to Equation (2.2) and notice that in simulating an NSPACE[logn] property, two universal quantifiers ranging from 1 to n are used. Their purpose is only to make a choice between the first half and the second half of the path. It makes sense to minimize the universal choices when simulating an existential class so we replace "$\forall u \forall v$" in Equation (2.2) by "$\forall b$", where b is boolean valued. Thus:

$$P_n(x,y) = \exists z \forall b \exists u \exists v \left([(b=0 \,\&\, u=x \,\&\, v=z) \text{ or } (b=1 \,\&\, u=z \,\&\, v=y)] \right.$$
$$\left. \&\quad \exists x \exists y [x=u \,\&\, y=v \,\&\, P_{n/2}(x,y)] \right) \qquad (2.4)$$

Define BUVar &Sz[v(n),z(n)] to be the family of properties expressible in v(n) variables and size O[z(n)], where the existential quantifiers still range from 1 to n, but the universal quantifiers are boolean. Thus it is easy to see that GAP is in BUVar &Sz[k,logn], and more generally,

Theorem 2.2: For $s(n) \geq \log(n)$,

$$\text{NSPACE}[s(n)] \subseteq \text{BUVar \&Sz}[k \cdot s(n)/\log(n), s(n)^2/\log(n)] .$$

Section 2.2: Relations of BUVar & Sz to other models.

Recall a definition and result of Sudborough [Sud78]:

Definition: AuxPDA[s(n),t(n)] is the class of languages accepted by a two way nondeterministic push down automaton with auxilliary work tape of size s(n), running in time t(n). For readability we will write AuxPDA[logn, n^k] to mean $\cup_{k=1,2,...}$ AuxPDA[logn, n^k].

Fact (Sudborough): AuxPDA[logn, n^k] = Log(CFL).

In [Ruz79a] Ruzzo defines an accepting computation tree of an alternating Turing machine M to be a tree whose root is a starting ID of M, whose nodes are intermediate ID's, and whose leaves are all accepting configurations. Each universal node, u, has all its possible next moves as offspring, while the existential nodes, e, lead to exactly one of e's possible next moves. We say that a language C is in ASPACE &TS[s(n),z(n)] if all members of C of size n are accepted in a computation tree using space s(n) and tree size, (number of nodes), z(n). Ruzzo relates this new measure to auxilliary pda's via,

Fact (Ruzzo): ASPACE &TS[s(n),z(n)^k] = AuxPDA[s(n),z(n)^k].

Notice that both the tree size model and the AuxPDA charge much more for universal moves than for existential ones. The following theorem shows that we get the same classes in our expressibility measure by restricting all universal quantifiers to be boolean. In a sense we charge log(n) times as much for a universal choice as for an existential.

Theorem 2.3: BUVar &Sz[k•v(n), z(n)] = ASPACE &TS[v(n)log(n), $2^{k•z(n)}$].

proof: ($\subseteq$): Note that for readability we have omitted the union over k for both sides of the above equation. Given an input structure G with n element universe we can generate F_n, the n^{th} sentence in our uniform sequence. We must show that in ASPACE &TS[v(n)log(n), $2^{k•z(n)}$] we can check if G$\models F_n$.

To test if G satisfies F_n we read the sentence from left to right holding the present values of variables $x_1 \ldots x_{v(n)}$ in our v(n)log(n) memory. Note that each non-boolean variable may have value 1 to n corresponding to an element of G. At existential quantifiers, $\exists x_i$, we existentially choose some x_i from the universe of G and at universal choices, $\forall b_j$, we universally choose b_j. When we come to atomic predicates, e.g. $E(x_1, x_2)$ or $b_{17} = 0$, we can check their truth because we have the current values of the variables. Note that this accepting procedure has tree size $2^{k \cdot v(n)}$ because we may make a binary universal split $k \cdot v(n)$ times.

($\supseteq$): Here we follow a proof of Ruzzo [Ruz79a]. We must express the property Accept(r,z) which means that the alternating Turing machine M will accept in tree size z when started with ID r. We express Accept(r,z) by choosing a point p in the middle of the tree whose subtree is of size between 1/3 and 2/3 of the original tree. Thus,

$$\text{Accept}(r,z) \;=\; \exists p \left(\text{Accept}(r,\langle p\rangle,(2/3)z) \;\&\; \text{Accept}(p,\langle\rangle,(2/3)z) \right)$$

Here Accept(r,$\langle q_1 \ldots q_k \rangle$,z) means that there is a computation tree of size z starting at r such that each leaf is either an accepting configuration or one of $q_1 \ldots q_k$.

Our only trouble is to insure that the list $\langle q_1 \ldots q_k \rangle$ stays of constant size. Whenever the list is of length three we take an extra move to split it in half by finding a point p above two of the three nodes in the list,

$$\text{Accept}(r,\langle q_1,q_2,q_3\rangle,z) \;=\; \exists p \left(\text{Accept}(r,\langle q_1,p\rangle,z) \;\&\; \text{Accept}(p,\langle q_2,q_3\rangle,z) \right)$$

Note that in the above we can add a boolean universal quantifier and use the abbreviation trick to write Accept(-) only once on the right. Also note that the above is a slight lie since we don't know which pair of q's p will be above. In fact we would have to say,

$$\exists p \, (\exists s_1 s_2 s_3 \text{, a permutation of } q_1 q_2 q_3) \left(\text{Accept}(r,\langle s_1,p\rangle,z) \;\&\; \text{Accept}(p,\langle s_2,s_3\rangle,z) \right)$$

Thus we can write Accept(-) with a constant number of ID's, i.e. v(n) variables, and the size of the sentence is O[logz]. This proves Theorem 2.3. ∎

Corollary 2.4:

a. $BUVar\ \&Sz[k,\log n] = Log(CFL)$

b. $Var[k] = Var\ \&Sz[k,n^k] = PTIME$

proof:

(a): From the above theorem, together with the results from Ruzzo and from Sudborough:

$$\begin{aligned} BUVar\ \&Sz[k,\log n] &= ASPACE\ \&TS[\log n, n^k] \\ &= AuxPDA[\log n, n^k] \\ &= Log(CFL)\ . \end{aligned}$$

(b): By our uniformity condition, (*), the n^{th} sentence of a Var[k] property can be generated in DSPACE[logn] and so it is of size at most n^k. By Theorem 2.3,

$$\begin{aligned} Var\ \&Sz[k,n^k] &= ASPACE\ \&TS[\log n, 2^{n^k}] \\ &= ASPACE[\log n] = PTIME\ . \end{aligned}$$

□

The above corollary rounds out a pleasing relationship between expressibility and computation. We have shown:

1. The size of a sentence needed to express condition C is polynomially related to the amount of memory space needed to check if C holds for an input, and,

2. Conditions which can be expressed with v variables are just those conditions which can be checked in $DTIME[n^{O(v)}]$.

Section 2.3: Relations to Alternating Space and Time.

The next theorem generalizes the above results giving a remarkably close relationship between expressibility and alternating machine complexity. Let $BVar\ \&Sz[v(n),z(n)]$ be those properties expressible in a sentence with v(n) *boolean* variables and size z(n). For each predicate symbol, E, from

τ, we add the predicate symbols, E_1, E_2 ... where $E_a(b_1 \ldots b_{\log n}, c_1 \ldots c_{\log n})$ means $F(b,c)$ where b has binary vertex number $b_1 \ldots b_{\log n}$.

Theorem 2.5: $ASPACE\&TIME[s(n),t(n)] = BVar\&Sz[k \cdot s(n), t(n)]$.

proof: ($\supseteq$): Given an input structure G with n element universe we can generate F_n the n^{th} element in our uniform sequence. We must show that in $ASPACE\&TIME[s(n),t(n)]$ we can check if $G \models F_n$.

To test if G satisfies F_n we read the sentence from left to right holding the present values of the variables $b_1 \ldots b_{s(n)}$ in our memory. At quantifiers $\exists b_i$ or $\forall b_i$ we make the appropriate existential or universal choice of a new value for b_i. Similarly at &'s (or "or"s) we universally (or existentially) choose one branch and proceed. We have G and the values of the variables so we may check the truth of atomic formulas: $b = 0$, or $F(b_1 \ldots c_{\log n})$, in number of steps a constant times their length.

($\subseteq$): Going the other way we must write the sentence, $Accept_t(r)$ meaning that alternating Turing machine M when started at instantaneous description r will reach acceptence in t steps. We accomplish this by saying that if r is existential then there exists some next step x and $Accept_{t-1}(x)$, whereas if r is universal then for all next steps x, $Accept_{t-1}(x)$.

A technical difficulty here is that if at each step we recopy the entire ID then the size of the resulting sentence will be $s(n)t(n)$. To simplify the problem let us assume that the moves of our Turing machine alternate at every step and branch into at most two moves. Thus we can write,

$$Accept[ID_0] = \exists b_1 \forall b_2 \ldots Q_s b_s \exists ID_1 \left(ID_0 \rightarrow b_1 \ldots b_s \rightarrow ID_1 \ \& \ Accept[ID_1] \right)$$

Here "$ID_0 \rightarrow b_1 \ldots b_s \rightarrow ID_1$" means that there is a computation whose i^{th} move makes the b_i^{th} choice and leads from ID_0 to ID_1 in s steps. This is a deterministic computation of length s and can be asserted to exist with $O[s]$ symbols. The details of how to simulate $DTIME[s]$ in $BSize[s]$ are given in Theorem 2.10. ∎

We have demonstrated an exact relationship between alternating Turing machines and quantified boolean formulas. If we return to the more natural language of the input structures, i.e. variables ranging from 1 to n, then the needed number of variables to simulate s(n) space becomes $s(n)/\log(n)$.

It is not clear, however, how to do better than size t(n) to simulate time t(n) because the machine might go through t(n) alternations. Thus we can only show:

Corollary 2.6: For $s(n) \geq \log(n)$,

$$\text{ASPACE\&TIME}[s(n),t(n)] \subseteq \text{Var\&Sz}[k \cdot s(n)/\log(n),\ t(n)] \subseteq \text{ASPACE\&TIME}[s(n),t(n)\log(n)] .$$

Let ASPACE &Alt[s(n),a(n)] be those problems accepted in alternating space s(n) with at most a(n) alternations between existential and universal states. Then:

Corollary 2.7: Let $s(n) \geq a(n)\log(n)$. Then:

$$\text{ASPACE\&Alt}[s(n),a(n)] \subseteq \text{Var\&Sz}[k \cdot s(n)/\log(n),\ (a(n) + s(n))s(n)/\log(n)] ,$$

and in particular,

$$\text{ASPACE\&Alt}[\log n,\log n] = \text{Var\&Sz}[k,\log n] .$$

proof: To assert the acceptence of an ASPACE &Alt[s(n),a(n)] computation we assert the existence of a(n) ID's where the alternations occur. Each path between the ID's has no alternations and so can be expressed in $\text{Var\&Sz}[k \cdot s(n)/\log(n),\ s(n)^2/\log(n)]$ by Theorem 2.1 . We write $\text{ACCEPT}_a(\text{ID}_0)$ to mean that ID_0 leads to acceptence in a alternations:

$$\text{ACCEPT}_{2a}(\text{ID}_0) = \exists\, \text{ID}_1 \left(\text{EPath}(\text{ID}_0,\text{ID}_1) \ \&\ \text{ACCEPT}_{2a-1}(\text{ID}_1) \right)$$

$$\text{ACCEPT}_{2a-1}(\text{ID}_1) = \forall\, \text{ID}_0 \left(\text{APath}(\text{ID}_1,\text{ID}_0) \Rightarrow \text{ACCEPT}_{2(a-1)}(\text{ID}_0) \right) .$$

Note that in the above EPath(x,y) and APath(x,y) are the $\text{Var\&Sz}[k \cdot s(n)/\log(n),\ s(n)^2/\log(n)]$ formulas which assert the existence of a computation path from x to y all of whose intermediate states are existential, respectively universal. As in the proof of Lemma 1.3 we can use the abbreviation trick to conglomerate the two terms on the right, making the size of $\text{Accept}_a(\cdot)$ equal to:

$$a \cdot s(n).\log(n) + \text{Size(EPath)}$$

$$= (a(n) + s(n))s(n)/\log(n) , \text{ as desired.}$$ ∎

The above corollary interested us especially because we now have natural classes, Log(CFL) and ASPACE &Alt[logn,logn] identified with both of the the intermediate terms in the following containment:

Corollary 2.8:

$$\text{NSPACE}[\log n] \subseteq \text{BUVar}\,\&\text{Sz}[k,\log n] \subseteq \text{Var}\,\&\text{Sz}[k,\log n] \subseteq \text{DSPACE}[\log^2(n)]$$

Corollary 2.8 which comes immediately from Theorems 2.1 and 2.2 sheds some light on the difference between $\text{DSPACE}[\log^2(n)]$ and $\text{NSPACE}[\log n]$. We conjecture that all three containments above are proper, but it is not even known that $\text{NSPACE}[\log n] \neq \text{DSPACE}[\log^2(n)]$.

Section 2.4: On Var[k].

We have shown in Corollary 2.4 that PTime = Var[k]. In this section we study this relationship in more detail. It would be lovely if Var[v] were exactly equal to $\text{DTIME}[n^v]$ but things are not quite that nice. The problem is with the size of the v variable sentence. We show in Theorem 2.9 that Var &Sz[v,z] is contained in $\text{DTIME}[z \cdot n^v \log(n)]$. Thus we would like a natural way (other than the uniformity condition) to limit the size of a v variable sentence, ideally to n^v.

On the other hand we will see that it makes sense for a sentence with v variables to be of size 2^{n^k}. Corollary 2.12 shows that it is possible to interpret Var[k] and ASPACE[logn] in an extended sense inwhich case they are both equal to PSPACE.

Theorem 2.9:

a. $\text{Var}\,\&\text{Sz}[v(n),z(n)] \subseteq \text{DTIME}[z(n)v(n)\log(n) \cdot n^{v(n)}]$

b. When v(n) is a constant we assume the syntactic uniformity condition (**) in the definition of Var &Sz. Then (a) can be improved to:
$\text{Var}\,\&\text{Sz}[v,z(n)] \subseteq \text{DSPACE}\,\&\text{TIME}[n^v, z(n)\log(n) \cdot n^v]$

proof: By the uniformity condition, given input G of size n we can generate F_n in time z(n). Check whether $G \models F_n$ as follows: Assume that all $\neg$'s have been pushed through to the inside and consider the parse tree for F_n.

Each of the v(n) variables may take on any of the n values of the universe of G. Starting at the leaves of the parse tree make a list of all the v(n)-tuples of assignments which make the given nodes true in G. We can pass up the tree toward the root computing the values making each node true as we go.

For example, an "&" node's list is gotten by intersecting the two lists it leads to, a "$\forall x_1$" node gets those tuples $<-, x_2 \ldots x_{v(n)}>$ which are in the preceding node's list with all values of x_1.

When we reach the root either our list will have all $n^{v(n)}$ possibilities or it will be empty, since F_n has no free variables. $G \models F_n$ if and only if we are in the former case.

On a RAM each of these steps can be done in time linear in the length of the list, i.e. $n^{v(n)}$. On an "indexing Turing machine", i.e. a machine which can write the location of a tape cell on an index tape and immediately see that cell's contents, we can thus acheive $\log(n)n^{v(n)}$. This is the time required to compute each node's list from the previous lists. Since there are at most z(n) nodes we have shown (a).

For (b) note that the uniformity condition (**) forces F_n to have an essentially linear parse tree. Thus only one list of size n^v need be remembered as we pass up the tree. ∎

The next theorem gives a converse relation of Var &Sz to DSPACE &TIME:

Theorem 2.10: DSPACE &TIME[s(n),t(n)] $\subseteq$ BVar &Sz[5log(s(n)),t(n)].

proof: We show how to describe a computation of Turing machine M running in DSPACE &TIME[s(n),t(n)] on input G. We will construct formulas, $C_t(p,x)$, meaning, "Symbol x occurs in cell p at time t." $C_t(p,x)$ will be written with 5log(s(n)) boolean variables.

The idea is to say that there exists a triple of cell values $x_{-1}\, x_0\, x_1$ in the previous move which lead to x in one move of M, and x_i occurs in cell p+i at time t-1. In symbols:

$$C_t(p,x) = \exists x_{-1}\, x_0\, x_1 \left(x_{-1}\, x_0\, x_1 \rightarrow 1 \rightarrow x \quad \& \quad \wedge_{i=-1\ldots 1} C_{t-1}(p+i,x_i) \right)$$

We can use the abbreviation trick to write C_{t-1} only once on the right. Note that although we write p and x as single variables they are really tuples of booleans of size log(s) and log(|M|) respectively. A

problem arises with how to express "p+i." To write "3q=p+i," requires writing log(s) booleans at each step, making the size of C_t, tlog(s). We can get around this by keeping a list of the added bits, i_1 ... $i_{log(s)}$ and once every log(s) steps writing, "3q=p+ i_1 + ... +$i_{log(s)}$." The latter sum can certainly be written in Size[$\log^2(n)$] <s <t symbols. It need only be written once in the whole sentence. Thus the total size is O[s]. The number of variables required is 3log(s) to store p, q, i_1 ... $i_{log(s)}$, plus log(s) + log(|M|) to store another p and x. Thus 5log(s) certainly suffices. We suspect that the constant 5 can be greatly reduced. ∎

Returning to variables ranging from 1 to n, and recalling the coding of log(n) bits into a single variable we can prove the following:

Corollary 2.11: There is a fixed constant c_0 such that for all $s(n) \geq \log(n)$,

$$\text{DSPACE \&TIME}[s(n),t(n)] \quad \subseteq \quad \text{Var \&Sz}[c_0 \log(s(n))/\log(n), t(n)]$$

It makes sense to consider a sentence with k variables which is of length greater than n^k. Similarly we can consider an ASPACE[logn] machine which runs for more than n^k steps. Generalize the definition of ASPACE &TIME[s(n),t(n)] to be the family of languages accepted by an ASPACE[s(n)] machine with a t(n) clock. Such a machine's accepting configuration is an accept state with the clock equal to 0, however the machine is never allowed to look at its clock. With this definition Theorems 2.5,2.9, and 2.10 make sense and are true for all t(n). It is now possible to ask, "What is ASPACE &TIME[s(n),t(n)] with $t(n) > 2^{s(n)}$?"

Corollary 2.12:

a. $\quad \cup_{k=1,2,\ldots} \text{ASPACE \&TIME}[\log n, n^k] \quad = \quad \cup_{k=1,2,\ldots} \text{Var \&Sz}[k, n^k] \quad = \quad \text{PTIME}$

b. $\quad \cup_{k=1,2,\ldots} \text{ASPACE \&TIME}[\log n, 2^{nk}] \quad = \quad \cup_{k=1,2,\ldots} \text{Var \&Sz}[k, 2^{nk}] \quad = \quad \text{PSPACE}$

Right now line (a) is what we call ASPACE[logn] and Var[k], but it is not clear whether or not line (b) deserves at least the latter name.

We have already shown line (a) in Corollary 2.4. The left hand equality of line (b) follows from Corollary 2.6. The two containments of the second equality follow from Theorems 2.9 and 2.10.

Chapter 3

Ehrenfeucht - Fraisse Games

In this chapter we will employ Ehrenfeucht-Fraisse games to obtain lower bounds for the quantifier measure. These games are due to Fraisse and Ehrenfeucht. (See [Fra54] or [Ehr61] for discussion and proof of Theorem 3.1 .) Two persons play the game on a pair of structures. Player I tries to demonstrate a difference between the two structures, while Player II tries to keep them looking the same.

In Section 3.1 the games are defined and an example is given. Section 3.2 includes two easy lower bound proofs using the games. Finally in Section 3.3 we prove that a certain PTIME property is not in SIZE[$w.o.\ Suc][\log^k(n)]$ for any k. This proof should probably be omitted on a first reading.

Section 3.1: Definition of the game.

Given two structures, G and H, of the same finite type, τ, we define the n move game on G and H as follows:

Player I chooses an element of G or H and Player II chooses a corresponding element from the other one. This is repeated n times. At move i, g_i and h_i, elements of G and H respectively, are chosen.

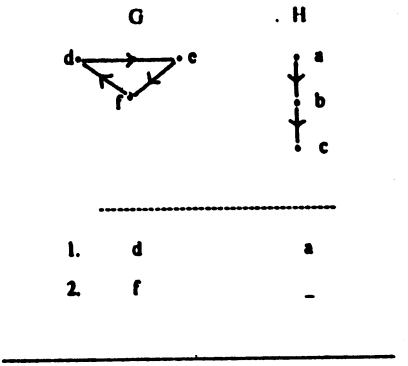
We say that Player II wins if the map f which takes the constants from G to the constants from H, and maps g_i to h_i, is an isomorphism of the induced substructures. (That is f preserves all of the symbols of τ. For example, if G and H are of type $\tau = \tau_g$, then $E(s,g_i)$ holds in G just if $E(s,h_i)$ holds in H.)

We say that two structures of type τ are n-equivalent if they satisfy all the same sentences in $L[\tau]$ of quantifier rank n. The fundamental fact about Ehrenfeucht-Fraisse games is:

Theorem 3.1 (Fraisse, Ehrenfeucht) : Player II has a winning strategy for the n move game on A and B if and only if A is n-equivalent to B.

We will prove a generalization of this result in Theorem 4.1 . For now we just give an example. Consider the graphs G and H of Figure 1. G has the property that each of its vertices has an edge leading to it, but this is not true of vertex a in H. Thus G and H disagree on the sentence, $S = \forall x \exists y E(x,y)$. By Theorem 3, Player I has a winning strategy for the game of length 2. Indeed, on the first move Player I chooses a. II must answer with a point from G, say d. Now I can pick f from G. II will lose because there is no point in H with an edge to a.

Figure 3.1: An Ehrenfeucht-Fraisse game.



Section 3.1: Some Easy Lower Bounds.

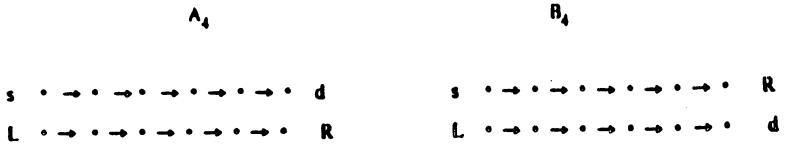
Recall that in Theorem 1 we showed an upper bound of Size[logn] for GAP. The following theorem proves that this is a lower bound as well. Note that we can express GAP in quantifier rank

exactly log(n), and so the Ehrenfeucht-Fraisse game is a tool fine enough to decide expressibility up to an additive constant!

Theorem 3.2: GAP is not expressible in quantifier rank log(n) -2.

proof: Fix n>4 and let m = (n-4)/2. We construct the graphs A_m,B_m as follows: Each graph consists of two lines of m + 2 vertices as in Figure 3.2. In both graphs s is the top left vertex; but, d is the top right vertex in A_m and the bottom right vertex in B_m. Thus A_m is in GAP, but B_m is not.

Figure 3.2: A_m and B_m.



We will now show that A_m is (logn -2)-equivalent to B_m. From this it follows that no sentence of quantifier rank logn -2 can express the property, "There is a path from s to d."

By Theorem 3.1 it suffices to show that Player II wins the logm move game on A_m ,B_m. Indeed, the following is a winning strategy for II:

If Player I plays the i^{th} vertex in some row of A (or B), II will always answer with the i^{th} vertex of one of the rows in B (or A). The initial constraint is that the endpoints s,d,L,R are answered by the similarly labelled endpoints. With k moves to go, if Player I chooses vertex x within 2^k steps of an endpoint (or previously chosen vertex, a_i), then II must answer with a vertex on the same row as the corresponding endpoint (or b_i), and at the same distance. If x is not within 2^k steps of such a point then II may answer with any point not within 2^k steps of an endpoint or chosen point.

A proof by induction will show that if II follows the above strategy for logm moves, then a conflict (i.e. two points on different rows, both within 2^k steps) will never arise. Thus Player II wins the logn -2 move game. ∎

Theorem 3.2 remains true for ordered graphs. The proof is similar, but the graphs require three rows each so that d is not the last vertex in B_m.

It is interesting to note that in the above case our measure does not distinguish between deterministic and nondeterministic space. The lower bound of O[logn] is shown for graphs with at most one edge leaving any vertex. The gap problem for such graphs, (called GAP1 and discussed in [HHM78] and [Jon75]), is in DSPACE[logn].

As promised we now show that $L[\tau_g]$, the language of graphs without Suc, is insufficient for describing all graph problems. Our counterexample consists of a totally disconnected graph. The same example could be built with connected graphs of unbounded degree. The idea is that the edge relation is of no use and so we must name all the points inorder to count them.

Proposition 3.3: EVEN, the set of graphs with an even number of vertices, is in DSPACE[logn], (in fact it's in DSPACE[0]), but it is not in Size(w.o. Suc)[h(n)] for any h(n) asympotically less than n.

proof: We already know by Theorem 1.2 that EVEN is in Size[logn]. To prove Proposition 3.3 let TD_n be the totally disconnected graph with n vertices. We show that TD_{n-1} is n-1 equivalent to TD_n. It follows that quantifier rank n is needed to express EVEN.

We only need to show that Player II wins the n-1 move game on TD_{n-1} and TD_n. Her obvious winning strategy is to match a chosen vertex with *any* vertex from the other side subject to the condition that a point chosen twice will be answered with the same point both times. Since the edge relation is always false in both structures, the resulting sequences of points are isomorphic. ∎

The proposition above concerns itself with the difference between Size(w.o. Suc) and Size . In the next section we will produce a more natural graph problem in P-Time, which is not in Size(w.o. Suc)[$\log^k$(n)]. The graphs there are connected and of bounded degree. We feel that the latter example concerns itself with time versus space.

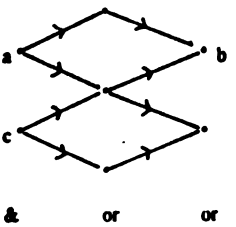
Section 3.3: P-TIME and the Size measure.

Let an alternating graph be a directed acyclic graph whose vertices are marked "&" or "or". Suppose that a and b are vertices of alternating graph G, and a has edges to $x_1 \ldots x_n$. We say that b is reachable from a iff:

1. a=b; or
2. a is marked "&", $n \geq 1$, and b is reachable from all the x_i's; or,
3. a is marked "or" and b is reachable from some x_i.

Note that if all vertices are marked "or" then this is the usual notion of reachability. (See Figure 3.3 where b is reachable from a, but not from c.) Note that we could generalize this definition to include infinite graphs or graphs with cycles by saying that "b reachable from a" is the *smallest* relation satisfying 1-3.

Figure 3.3: An Alternating Graph



Now define AGAP to be the set of alternating graphs inwhich d is reachable from s.

Proposition 3.4: AGAP is complete for polynomial time with respect to log-space reducibility.

proof: To see if G is in AGAP we start at d and proceeding backwards mark all the points from which d is reachable.

A detailed proof of completeness is omitted; the idea is that AGAP is complete in a natural way for alternating log space, which is known to be equivalent to P-TIME. (See [ChST76] or [Koz76].)

Boolean circuit value problems which are very similar have previously been shown to be complete for P. See for example [Gol77]. ∎

We must now add the predicate A(x) meaning that vertex x is marked "&". Let $\tau_{ag} = \langle E,A,S,D \rangle$, be the type of alternating graphs. Our next theorem shows that in $I[\tau_{ag}]$ the polynomial time property AGAP is not expressible with quantifier rank $\log^k(n)$. If this went through with the addition of successor then we would have shown that P is not contained in $SPACE[\log^k(n)]$.

Theorem 3.5: Let f(n) be any function that is asymptotically less than $2^{(\log n)^{1/2}}$. Then AGAP is not in Size(w.o. Suc)[f(n)]. In particular, AGAP is not in Size(w.o. Suc)[$\log^k(n)$] for any k.

proof: For all sufficiently large m, we produce graphs G_m and H_m with the following properties:

1. $|G_m| = |H_m| = n$, and $n < m^{1+\log m}$. Thus $\log(n) < \log(m)(\log(m)+1)$, and $2^{(1+\log n)^{1/2}} < m$.
2. G_m is m-equivalent to H_m.
3. G_m is in AGAP, but H_m is not.

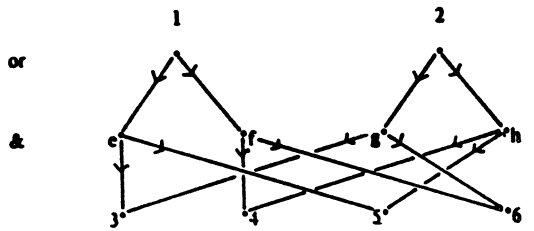
When these conditions are met we will have shown that anything less than quantifier rank $2^{(\log n)^{1/2}}$ does not suffice to express the Alternating Graph Accessibility Problem without successor.

The first step is to introduce the building block out of which G_m and H_m will be constructed:

Lemma 3.5a: Let X be the alternating graph pictured in Figure 3.4. Then X has automorphisms f, g, and h, with the following properties:

1.	f	switches	3 & 4	and	1 & 2,	leaving	5 & 6 fixed.
2.	g	switches	1 & 2	and	5 & 6,	leaving	3 & 4 fixed.
3.	h	switches	3 & 4	and	5 & 6,	leaving	1 & 2 fixed.

Figure 3.4: Switch X



proof: The idea is that when X is placed in our graphs each pair, 1,2 3,4 5,6 will consist of one point which can reach d and one which cannot. Think of points which can reach d as "true," and those which cannot as "false." Then in symbolic notation:

1 = [(e) or (f)] = [(3 & 5) or (4 & 6)]

2 = [(g) or (h)] = [(3 & 6) or (4 & 5)]

The proof of the lemma is an easy computation. ∎

We will say that a pair u,v is off if u is true and v is false. If u is false and v is true then the pair is on. Thus, X is a switch whose top pair is on just if exactly one of its bottom pairs is on.

The proof of Theorem 3.5 proceeds as follows: First we produce exponentially large graphs P_m and Q_m built up from switch X. P_m and Q_m differ on the AGAP property but we will see in Lemma 3.5b that they are m-equivalent. The final and most technical part of the proof is to reduce P_m and Q_m to G_m and H_m, graphs of size about $m^{\log m}$ which retain the above properties

Figure 3.5: P_m (if s=A), Q_m (if s=B)

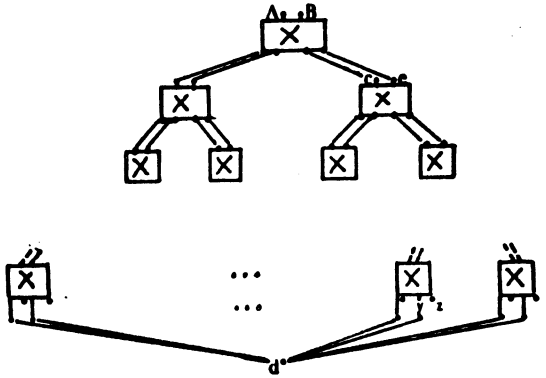


Figure 3.5 shows $2^{1+2m\log(2m)}-1$ copies of the switch X, arranged in a binary tree. Let P_m be the graph pictured in Figure 3.5, with s=A. Let Q_m be the same graph, but with s=B. Thus P_m is in AGAP while Q_m is not. However,

Lemma 3.5b: P_m is m-equivalent to Q_m.

proof: We will show that Player II wins the m length game on P_m and Q_m. One way to express the difference between P_m and Q_m is to say that they are the same except that the top pair in Q_m is switched. Another way of thinking of it is that in Q_m one of the bottom pairs, for example y,z, is switched. That is in P_m, y is connected to d, but in Q_m z is connected to d. X has the property that switching one pair on the bottom will result in the top pair being switched.

The idea behind Player II's winning strategy is that the difference between P_m and Q_m could be removed by switching any of the $2^{2m\log(2m)}$ pairs on the bottom row. With only m moves, Player I cannot eliminate all of these possibilities.

To simplify the proof let us first consider a different game. Let $T_{2m\log(2m)}$ be the binary tree of height 2mlog(2m). This is a schematic version of P_m and Q_m where each point represents the switch, X, and each line represents a pair of lines.

We play a modified Ehrenfeucht-Fraisse game on $T_{2m\log(2m)}$, call it the "on-off" game. On each move of this new game, Player I picks a point and Player II must answer "on" or "off". Player II must

also obey the rules that the top vertex, if chosen, is on, and any chosen vertex on the bottom is off. (Intuitively "off" corresponds to matching the top left vertex of the chosen switch in P_m to the same vertex in Q_m ; "on" means matching it to the top right vertex.) We say that Player II wins if for any triple of chosen points, L,M,N , such that M and N are the two offspring of L, L is on iff exactly one of M and N is on. This rule captures the behavior of the switch X.

Lemma 3.5c: Suppose that each vertex in row r of T_n is labelled on or off. Then any 2^k -1 points on or below row r+k may be labelled in any self-consistent fashion and there will still be a labelling of the rest of the graph which generates row r.

proof: By "self-consistent" we mean that with the labelling of row r removed, the labelling of the 2^k -1 points may be extended to a consistent labelling of the entire tree. The proof is by induction on k:

If k = 1 then no mater which point is chosen we are free to label its sibling as we please inorder to give the desired label to its parent.

Inductively suppose that 2^k -1 points are labelled on or below row r+k. Let L be the set of left offspring in row r+1, R the set of right offspring. Clearly at most one of these sets, say L, has more than 2^{k-1} -1 of its descendants labelled. Label all of the vertices in L in any consistent fashion. Now by induction we may label the points in R as we choose. Thus we may label row r as desired. ∎

It follows that Player II wins the 2m-move on-off game on $T_{2m\log(2m)}$. Her strategy is to answer "off" whenever possible. Lemma 3.5c shows that no point can be forced on in fewer than m moves unless a point less than logm rows above is labelled on. Thus 2m moves do not suffice to force on a point in row 2mlog(2m).

We can now play the original m-move Ehrenfeucht-Fraisse game as follows: (See Figure 3.5.) When Player I chooses a point, for example c in P_m , II moves according to the strategy for the on-off game. If the point corresponding to c's switch is declared "off", then II answers c, if "on", then c, the opposite point in the pair. If a point inside a switch is chosen then II may simulate the moves of the on-off game for the switch's two descendants. If either of these descendants is "on" then the moves induce one of the automorphisms of switch X listed in Lemma 3.5a. Player II should perform this automorphism on the switch in question and answer accordingly.

We claim that this is a winning strategy for Player II, i.e. there is an isomorphism between the

chosen points from the two graphs. The rule that in the on-off game Player II must call the top point on and the bottom points off assures that s will be answered by s and any point touching d will be answered by a point also touching d. The fact that Player II wins the on-off game indicates that any triple of neighboring switches is matched up correctly. This proves Lemma 3.5b. ∎

The final step of the proof is to introduce the graph $D_{log m}$ to replace the binary tree in the above construction. $D_{log m}$ has about $m^{log m}$ vertices above row m but still has the property that no point in block k can be forced on before the k^{th} move. We define D_k below algebraically, but please refer to Figures 3.6, 3.7, and 3.8, which show portions of D_1, D_2, and D_3 respectively.

Vertices(D_k) =

$$\{ \langle x_1 \ldots x_k, r \rangle \mid r = b \cdot k + p,\ p < k,\ \ 0 \le x_i \le b+1 \ \text{ for } 1 \le i \le p \ \ \&\ \ 0 \le x_i \le b \ \text{ for } p < i \le k \ \ \}$$

Edges(D_k) =

$$\{(\langle x_1 \ldots x_k, r \rangle, \langle x_1 \ldots x_k, r+1 \rangle) \mid r \ge 0\} \ \cup$$
$$\{(\langle x_1 \ldots x_k, r \rangle, \langle x_1 \ldots x_{p-1}, x_p + 1, \ldots x_k, r+1 \rangle) \mid r = p \bmod (k)\}$$

Thus the vertices are k dimensional vectors and each row stretches the range of one of these dimensions by one. These graphs have k degrees of freedom, allowing us to prove:

Lemma 3.5d: Suppose that row r of D_k is entirely labelled. Then any $2^k - 1$ points on or below row r+k may be labelled in any self-consistent fashion and there will still be a labelling of the rest of the graph consistent with row r.

proof: We break the proof into two parts. First assume that the $2^k - 1$ chosen points lie on row r+k. The proof is by induction on k.

If k = 1 then we must show that any one point may be chosen in row r+1 without affecting row r. This is true because any configuration in row r is generated by a configuration in row r+1 and by its complement. Clearly one of these marks the chosen point correctly.

Let the j^{th} block of D_k be the k consecutive rows numbered j·k to j·k + k-1. Note that in passing from the i^{th} row of one block to the i^{th} row of the next block the range of each coordinate is stretched by 1. Assume for convenience that row r is at the bottom of block j.

Figure 3.6: Four Blocks of D_1

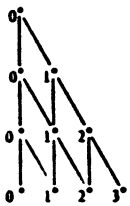


Figure 3.7: Four Blocks of D_2

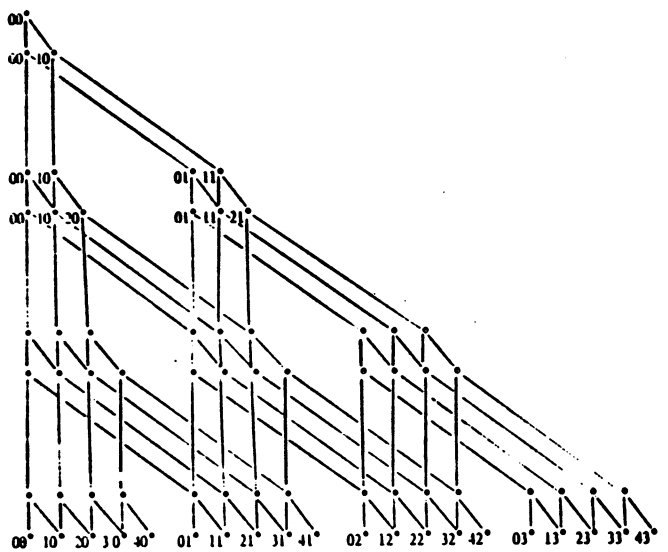


Figure 3.8: Three Blocks of D_3

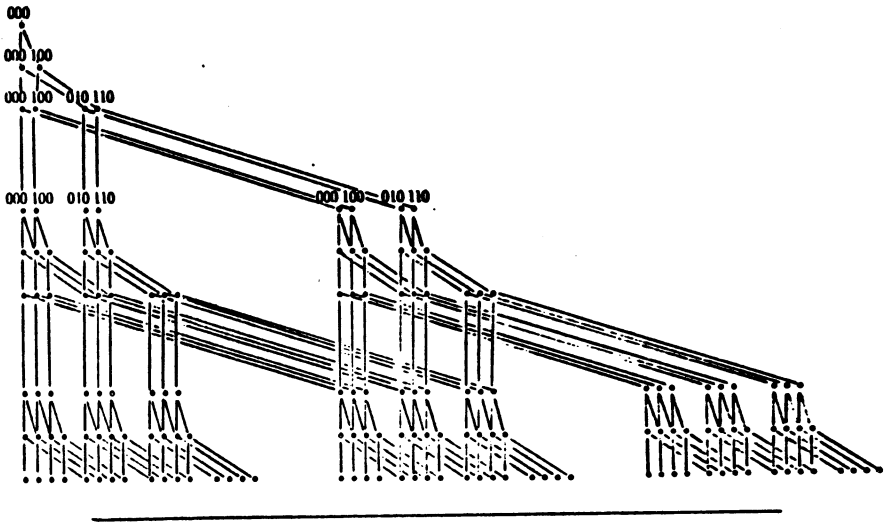
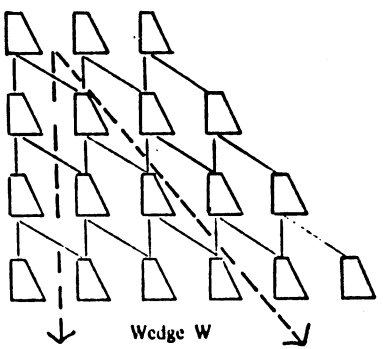


Figure 3.9: Schematic view of Wedge W.

Column: $i_0 - 1$ i_0 $i_0 + 1$



Row	Block
r	j
r+1	j+1
r+k	j+1
	j+2
	j+3

Let the i^{th} column of D_k be those points with k^{th} coordinate equal to i. Note that the i^{th} column of D_k is a copy of D_{k-1} with every k^{th} row repeated. In particular the i^{th} column of the j^{th} block of D_k is a copy of the j^{th} block plus the first row of the $j+1^{st}$ block of D_{k-1}.

Inductively consider any labelling of row r of D_k together with a choice of 2^k-1 labelled points on row r+k. Clearly at most one of the columns of row r+k has 2^{k-1} chosen points. Let this i_0^{th} column of row r+1 be chosen in any consistent fashion. All the other columns have at most $2^{k-1}-1$ chosen points in row r+k. Thus by induction all the other columns of row r+1 may be chosen as we please. Thus we can counteract the i_0^{th} column and choose row r as we please: Choose column i_0-1, row r+1 to be the sum of the desired column i_0-1, row r, and of column i_0, row r+1. Next choose columns i_0+1, i_0-2, and so on.

To complete the full lemma we must generalize our inductive assumption:

Claim: Suppose that row r of D_k is entirely labelled. Further assume that some of the edges are marked "¬" meaning that the signal going through that edge is reversed. Then any 2^k-1 points on or below row r+k may be labelled in any consistent fashion and there will still be a labelling of the entire graph giving the desired row r.

proof: As above at most one column of D_k, say i_0, has at least 2^{k-1} chosen points. Consider the wedge, W, above the i_0^{th} column of row r+1. W consists of column i_0 block j+1, columns i_0-i_0+1 block j+2, and so on. See Figure 3.9 for a schematic view of W.

Label the points of W in any fashion consistent with the labelling of the 2^k-1 chosen points, taking into account the edges marked "¬". Now consider column i_0-1. For each edge e from a to b in column i_0-1 it may be the case that there is also an edge from c to b where c is a point in W labelled "on". If so we mark e "¬" because the value of b will be the opposite of the value of a.

Next merge the repeated pair of rows in each block of column i_0-1, thus making a true copy of D_{k-1}. By our inductive assumption column i_0-1 with the added "¬"s may be filled in as we please. Similarly we can use the diagonals to the right of W to fill in columns i_0+1, i_0+2, ... of row r+1. Thus we can generate row r as we please. ∎

It follows from Lemma 3.5d that Player II wins the 2^k move on-off game on the first 2^k blocks of D_k. Her strategy is to say "off" until a point is forced on. No point p can be forced on unless more than 2^k-1 points have been chosen or there is a point marked on within k rows of p. Thus 2^k moves do

not suffice to force on a point in the bottom row.

Let G_m and H_m be the graphs arising from the first 2m blocks of $D_{\log(2m)}$ by replacing vertices by the switch X, just as P_m and Q_m arose from the binary tree of height 2mlog(2m). As before we let s be the top left point of G_m and the top right point of H_m. Thus G_m is in AGAP and H_m is not in AGAP.

Our above remarks imply that Player II wins the 2m move on-off game on $D_{\log(2m)}$. Thus, as in the proof of Lemma 3.5b, G_m is m-equivalent to H_m. This proves Theorem 3.5. ∎

Theorem 3.5 does not go through if we add "Suc". In the log(n) move game on numbered graphs if Player I chooses vertex i in A, then II must respond with vertex i in B. If Player II answers differently, then in the remaining moves Player I can keep cutting the successor path from the initial point to vertex i in half, thus exposing that this path is not the same length on the left as on the right. Clearly if G and H are not identical graphs there must be a pair of indices i,j such that there is an edge from v_i to v_j in one of the graphs but not the other. Thus Player I wins the log(n)+1 move game on G and H. His strategy is to play vertices v_i and v_j of G on the first two moves. As we have seen Player II is forced to answer with v_i and v_j from H. Now she has lost because the map between the first two elements is already not an isomorphism.

Thus two numbered graphs of size n are log(n)+1 equivalent only if they are identical. This is as expected because a pair of graphs G,H is indistinguishable to all log space Turing machines only if G=H.

Sometime after proving Theorem 3.5 we discovered to our surprise that with Suc we probably can write a sentence of length O[logn] which distinguishes G_m from H_m. This is done as follows: In a numbered graph a pair of vertices is endowed with an orientation. Thus a numbered copy of switch X is either right (orientation preserved) or wrong (orientation of the top pair is switched). Thus given a numbered graph which is either P_m or Q_m we can tell which by adding up the number of wrong switches and seeing if it is odd or even.

This does not quite work for distinguishing G_m from H_m because some of the switches in D_k have no effect -- that is their signals lead to the top an even number of times. (See for example vertex 10 in row 3 of Figure 3.6.) We believe (although we haven't written out the details) that the pattern of which vertices count is simple enough to admit expression via a log(n) quantifier formula C(x). If so then G_m can be distinguished from H_m with log(n) quantifiers by adding up the number of wrong switches y such that C(y).

To alleviate this problem we can replace the switch X in the above construction with a switch with m points. Thus to remember its orientation requires m bits rather than one. As above we can build graphs G_n' and H_n' which are $2^{(\log n)^{1/2}}$ equivalent without successor. We conjecture that even with Suc they are indistinguishable.

Chapter 4

Alternating Pebbling Games

In this chapter we present a new pebbling game to obtain lower bounds for Var &Sz(w.o. Suc). This game is a modification of Ehrenfeucht-Fraisse games discussed in Chapter 3. In Section 4.1 we define the alternating pebbling game and give an example. We will see in Theorem 4.1 that Player II wins the p pebble, m move game on structures G and H if and only if G and H agree on all sentences with p distinct variables and quantifier rank m. This theorem is proved in Section 4.2

In Section 4.3 we use the pebbling games to prove lower bounds for Var(w.o. Suc)[k]. We show that without a successor relation the Hamilton Circuit property and the graph isomorphism property cannot be expressed with only a constant number of variables. Furthermore the existence of a clique of size k as a subgraph of a given graph cannot be expressed with k-1 variables. (Of course k variables do suffice).

These results give further evidence (although they do not prove) that the above problems are not in PTIME.

Section 4.1: Definitions and Examples.

We now define the alternating pebbling game on G and H. The idea is that Player I will place pebbles on vertices in G or H and Player II will answer by pebbling corresponding points from the other structure. Player II will win if in the m moves of the game a difference is exposed between G and H. Essentially we have a modification of the Ehrenfeucht-Fraisse game inwhich only k points may be remembered at once rather than all m points in the m move Ehrenfeucht-Fraisse game.

Definition: The p pebble, m move game on G and H is defined as follows: Initially the pebbles, $g_1 \ldots g_p$, $h_1 \ldots h_p$, are off the board. On move i, Player I picks up a pebble g_j (or h_j), $1 \leq j \leq p$, and places it on a vertex of G (or H). Player II answers by placing h_j (or g_j) on a corresponding point of H (or G). Let $g_j(i)$ be the point on which g_j is sitting just after move i. After each move i, $0 \leq i \leq m$, define the map f_i as follows:

$$f_i : c^G \rightarrow c^H. \qquad g_j(i) \rightarrow h_j(i)$$

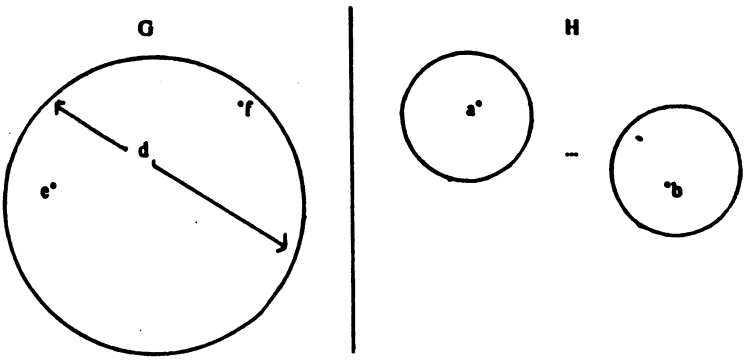
The map f_i takes the constants in G to the constants in H, and chosen points in G to the respective chosen points in H. We say that Player II wins if for each i, $0 \leq i \leq m$, f_i is an isomorphism of the induced substructures.

Theorem 4.1: Player II has a winning strategy for the p,m game on G,H if and only if G and H agree on all sentences with p variables and quantifier rank m.

We will give the proof, a minor modification of proofs in [Fra54] and [Ehr61], in the next section. First we will give an example. Consider the 4 pebble, d+1 move game on undirected graphs G and H where H is disconnected while G is connected with diameter d.

Player I wins the game as follows: On the first two moves he puts pebbles h_2, h_3 on vertices a,b such that a and b are in distinct components of H. Player II must place g_2, g_3 on some vertices e,f from G. There is a path of length at most d from e to f. Player I now uses the next d-1 moves to walk along this path with pebbles g_0 and g_1. Player II must answer with a path in H starting at a, and thus never reaching b. Thus at move d+1, two pebbles will coincide in G but not in H and Player I wins.

Figure 4.1: The 4, d+1 game on G and H.



Notice that Player I's strategy was to follow the following sentence, true in G but not in H: (Let $M(u,v) \equiv E(u,v)$ or $u=v$.)

$$\text{Diam}(d) = \forall x_2 \forall x_3 \exists x_0 \left(M(x_2,x_0) \ \& \ \exists x_1 \left(M(x_0,x_1) \ \& \ \exists x_0 \left(M(x_1,x_0) \& \ldots \& \exists x_{d+1} x_i \left(M(x_{i-1},x_i) \& M(x_i,x_3) \right) \ldots \right) \right) \right)$$

Also note that there is a sentence equivalent to Diam(d) whith only 3 variables and log(d)+1 quantifier depth which Player I would have played had he known about it.

Section 4.2: Proof of Theorem 4.1.

proof: ($\Rightarrow$): Suppose there is a sentence S with p variables and quantifier rank m such that G satisfies S but H does not. We must show that Player I wins the p pebble, m move game on G and H. This is proved by induction on m:

base case: If m = 0 then G and H differ on a quantifier free sentence, i.e. the constants in G satisfy a formula that the constants in H do not. Thus they are not isomorphic so Player I wins the 0 move game.

inductive step: If S is of the form $\neg A$, or $A \& B$, then G and H must disagree on one of A or B. Thus we may assume that S is of the form $\exists x_1 M(x_1)$. Here Player I places pebble g_1 on some vertex $g_1(1)$ from G so that $G \models M(g_1(1))$. No matter what II answers we will have $H \models \neg M(h_1(1))$. Thus by induction Player I will win.

Note that in the inductive step we have placed pebble g_1 so we must consider what happens if we later need g_1 again. The answer is that in $M(g_1(1))$, the substitution of $g_1(1)$ is made for all *free* occurrences of x_1. If later on in the game we need to place g_1 again it will be for some sentence $S' = Qx_1 N(x_1)$. Inside S' all occurrence of x_1 are bound by Qx_1, thus $g_1(1)$ does not occur and pebble g_1 may be safely reused.

($\leftarrow$): Conversely suppose that G and H agree on all sentences with p variables and quantifier rank m. We must show that Player II wins the p pebble, m move game on G and H. To facilitate an inductive proof we must slightly strengthen our claim. We prove the following:

Claim: Let $k \leq p$ and supose that $\langle G, c_1^G \ldots c_k^G \rangle$ and $\langle H, c_1^H \ldots c_k^H \rangle$ agree on all sentences S with new constants $c_1 \ldots c_k$, variables $x_1 \ldots x_p$, quantifier rank m, and such that nowhere in S does c_i occur within the scope of a quantifier for x_i. Then Player II has a win for the p pebble, m move game on G and H when started with the first k pebbles on $c_1^G \ldots c_k^G$ and $c_1^H \ldots c_k^H$ respectively.

Note that with $k = 0$ the claim reduces to what we need to show.

We prove the claim by induction on m. For $m = 0$ the map from the constants in G to the constants in H and $c_1^G \ldots c_k^G$ to $c_1^H \ldots c_k^H$ must be an isomorphism or else there would be a quantifier free sentence on which the two structures disagree.

For the inductive step asume that the premise of the claim holds and let Player I move placing, let us say, pebble g_1 on $g_1(1)$.

Consider the (finite) collection of sentences $S_1(x_1) \ldots S_r(x_1)$, in the language of G together with constants $c_2 \ldots c_k$ such that $\langle G, c_2^G \ldots c_k^G \rangle \models S_i(g_1(1))$. Let $S \equiv \exists x_1 \left(\bigwedge_{i=1 \ldots r} S_i(x_1) \right)$. Thus $\langle G, c_2^G \ldots c_k^G \rangle \models S$.

Thus by our assumption $\langle H, c_2^H \ldots c_k^H \rangle$ also satisfies S. Let $h_1(1)$ be a witness for x_1 in S. Now

$\langle G, g_1(1), c_2^G \ldots c_k^G \rangle$ and $\langle H, h_1(1), c_2^H \ldots c_k^H \rangle$ agree on all sentences, R, of quantifier rank m-1 and variables $x_1 \ldots x_k$ such that no c_i occurs within the scope of a quantifier for x_i. This is because any such $R(c_1)$ satisfied by G would be an S_i above and therefore also satisfied by H.

Our inductive asumption now shows that Player II wins the remaining m-1 moves of the game, proving the claim. This proves Theorem 4.1. ∎

Before we apply Theorem 4.1 it is useful to give a slightly different characterization of $G \equiv_{Var[k]} H$. What does it mean when G and H agree on all k variable sentences without successor? The idea is that if Player I chooses any r-tuple of points from G, $r \leq k$, then there is a corresponding isomorphic r-tuple from H. Furthermore if Player I adds a point to the tuple in G, and $r<k$, then there is a corresponding point in H which may be added preserving the isomorphism.

We have thus deduced the existence of a relation R on pairs of r-tuples from G and r-tuples from H, i.e. $R \subseteq \bigcup_{r=0\ldots k} G^k \times H^k$, satisfying:

a. $R(\langle\rangle,\langle\rangle)$

b. $R(g,h) \Rightarrow g \simeq h$

c. $(R(g,h) \;\&\; |g|<k) \Rightarrow \left(\forall x \in G\ \exists y \in H\ R(\langle g,x\rangle,\langle h,y\rangle)\right) \;\&\; \left(\forall y \in H\ \exists x \in G\ R(\langle g,x\rangle,\langle h,y\rangle)\right)$

d. If $g = \langle g_1 \ldots g_r \rangle$, let $\hat{g}_i = \langle g_1 \ldots g_{i-1}, g_{i+1} \ldots g_r \rangle$ be the r-1 tuple with g_i removed. Then $R(g,h) \Rightarrow R(\hat{g}_i,\hat{h}_i) \quad i = 1, \ldots, r$

Proposition 2.2: $G \equiv_{Var[k]} H$ if and only if there exists a relation R satisfying (a - d) above.

proof: It should be clear that R corresponds to Player II's winning strategy in the k pebble game on G and H. Thus if such an R exists then Player II can always win by matching chosen r-tuples in G with R-related r-tuples in H. Assume $R(\langle g_1(s) \ldots g_k(s)\rangle, \langle h_1(s) \ldots h_k(s)\rangle)$, i.e. the chosen points after move s are R-related. Think of Player I's moving of pebble g_i as two actions. First he picks up g_i. By (d) we know $R(\hat{g}_i(s),\hat{h}_i(s))$. Next he places g_i back on some new point $g_i(s+1)$. By (c) there exists y in H preserving the relation, i.e. with $h_i(s+1) = y$, $R(g(s+1),h(s+1))$. In particular $g(s+1)$ and $h(s+1)$ are isomorphic, and Player II wins.

Conversely, if $G \equiv_{Var[k]} H$ then define R from Player II's winning strategy as follows:

$$R = \{ (\langle x_1 \dots x_r \rangle, \langle y_1 \dots y_r \rangle) \mid \text{The k pebble game on G and H started with } g_i(0)=x_i, h_i(0)=y_i \ \ i=1, \dots, r, \text{ is a forced win for Player II} \}$$

The fact that Player II has a winning strategy for the k pebble game on G and H gives us (a). (b), (c), and (d) follow from the rules of the game. ∎

Section 4.3: Lower Bounds for Var(w.o. Suc)[k].

Following [Fag76] and [BlHa79], we write certain axioms for graphs. First:

$$T_0 = \forall x \forall y \left(\neg E(x,x) \ \& \ [E(x,y) \Rightarrow E(y,x)] \right)$$

T_0 says that G is loop free and undirected. We will assume in this section that all graphs satisfy T_0.

Fix k and let $1 \leq j \leq k-1$. The following sentences, $S_{k,j}$, say that for any choice of distinct vertices, $x_1 \dots x_j$ and $x_{j+1} \dots x_{k-1}$, there exists a vertex y different from the x_i's with an edge to every vertex in the first group and no edge to the second group.

$$S_{k,j} = \forall x_1 \dots \forall x_{k-1} \left(\left(\bigwedge_{0<r<k} x_i \neq x_r \right) \Rightarrow \exists y \left[\bigwedge_{0<i<j+1} E(y,x_i) \ \& \ \bigwedge_{j<i<k} (y \neq x_i \ \& \ \neg E(y,x_i)) \right] \right)$$

We use the $S_{k,j}$'s to write T_k, an axiom which says that every conceivable extension of a configuration of k-1 points to a configuration of k points is realizable.

$$T_k = \bigwedge_{0<j<k} S_{k,j}$$

A counting argument shows that almost all graphs satisfy T_k. Define $P_n(S)$, the probability that a graph of size n satisfies a sentence S, as follows:

$$P_n(S) = \#\{ G \mid G \models S, \text{ G a graph of size n} \} \ / \ \#\{ G \mid \text{G a graph of size n} \}$$

Theorem 4.3 ([Fag76], [BlHa79]): For any fixed $k>0$, $\lim_{n \to \infty} P_n(T_k) = 1$.

proof: Given k, and distinct vertices $x_1 \ldots x_{k-1}$ what is the probability that a random vertex y is a witness for $S_{k,j}$? It's just the probability that the $k-1$ possible edges $E(x_i, y)$ are correctly present or absent, i.e. $1/2^{k-1}$.

Thus the probability that none of a random $n-(k-1)$ vertices is a witness for $S_{k,j}$ is:

$$\left(1-(1/2^{k-1})\right)^{n-(k-1)} \qquad \leq \alpha^n .$$

The probability that any of the fewer than n^k sequences, $x_1 \ldots x_{k-1}, j$, cause T_k to fail is less than

$$n^k \cdot \alpha^k$$

and this last probability goes to 0 as n goes to infinity. ∎

We are interested in T_k because of the next result:

Theorem 4.4: For any two graphs G and H, $(G \models T_k \ \& \ H \models T_k) \Rightarrow \quad G \equiv_{Var[k]} H$.

proof: T_k says that every $k-1$ tuple may be extended to a k tuple in any conceivable way. It follows that the relation:

$$R \quad = \quad \{(\langle a_1 \ldots a_r \rangle, \langle b_1 \ldots b_r \rangle) \mid 0 \leq r \leq k,\ a_i \in G,\ b_i \in H, \text{ and } \langle a_1 \ldots a_k \rangle \simeq \langle b_1 \ldots b_k \rangle\}$$

satisfies (a) - (d) of Proposition 4.3. Therefore $G \equiv_{Var[k]} H$. ∎

Corollary 4.5: Graph Isomorphism is not in Var(w.o. Suc)[k].

proof: If GraphIso were in Var(w.o. Suc)[k] then there would be sentences $F_1, F_2 \ldots$ with k variables each such that for graphs G and H of size n,

$$\langle G,H \rangle \models F_n \qquad \leftrightarrow \qquad G \simeq H$$

By Theorem 4.1 there exist two non-isomorphic graphs G_k and H_k both satisfying T_k. Clearly $\langle G_k, G_k \rangle \models F_n$. But by Theorem 4.2, $G_k \equiv_{Var[k]} H_k$. It follows that Player II wins the k pebble game on $\langle G_k, H_k \rangle$ and $\langle G_k, G_k \rangle$. Her strategy is to answer points in the first component with the same point in the other copy of G_k, and to use Player II's winning strategy for the k pebble game on G_k and

H_k to answer moves in the right component. Thus,

$$\langle G_k, H_k \rangle \models F_n \text{ , but } G_k \text{ is not isomorphic to } H_k \text{ .}$$

This contradiction proves the corollary. ∎

Almost all graphs have a Hamilton circuit; however, in [BlHa79] it is shown that for any k there is a graph H_k which satisfies T_k and yet has no Hamilton circuit. It follows that there exist two graphs, G_k, H_k, both satisfying T_k and yet differing on the property of having a Hamilton circuit. Thus:

Theorem 4.6: "Hamilton Circuit" is not in Var(w.o. Suc)[k], for any k.

Using similar techniques we can show the following:

Theorem 4.7: Clique(k + 1) is not in Var(w.o. Suc)[k].

proof: Recall that Clique(k + 1) is the set of graphs with a complete subgraph of size k + 1. Clearly any graph satisfying T_{k+1} is in Clique(k + 1). We show that there exists a graph $H_k \models T_k$ such that H_k has no k + 1 clique. Define the graph $\Lambda_n = (V_n, E_n)$ as follows:

$$V_n = \{ \langle i, j \rangle \mid 1 \leq i \leq k,\ 1 \leq j \leq n \}$$
$$E_n = \{ (\langle i_1, j_1 \rangle, \langle i_2, j_2 \rangle) \mid i_1 \neq i_2 \}$$

Notice that Λ_n has no k + 1 clique because any set of k + 1 vertices will have two with the same first coordinate.

Let $\Lambda_n' = (V_n, E_n')$ be a random subgraph of Λ_n, i.e. each edge of E_n has probability 1/2 of being in E_n'. Now $\lim_{n \to \infty} \text{Prob}(\Lambda_n' \models T_k) = 1$. (This follows from the same argument as in the proof of Theorem 4.1, noting that every k-1 tuple from V_n has n points potentially satisfying T_k.) Let H_n be such a random Λ_n'. Thus H_n satisfies T_k but has no k + 1 clique. ∎

In a sense the above lower bounds seem too easy -- they may have more to do with how little can be said without successor than with why P≠NP. We feel, however, that it would be very interesting to determine precisely which problems are in Var(w.o. Suc)[k]. In chapter 6 we will touch upon some ideas concerning lower bounds for expressibility with successor, or approximations there of.

Chapter 5

Reductions and Complete Sets

In this chapter we define a reduction that honors the expressibility measure Var&Sz. Our definition comes from the notion of Interpretations Between Theories. The idea is that problem A is reducible to problem B if there is a uniform translation of all instances of A to instances of B. A similar reduction for generalized spectra is studied in [Gal.o77].

Section 5.1: Interpretations Between Theories.

We first give a short discussion of Interpretations Between Theories. For more detail one might read Section 2.1 of [End61]. Given similarity types τ_1 and τ_2 it is often possible to translate structures of type τ_1 into structures of type τ_2. Suppose $\tau_2 = \langle R_1 \dots R_p \rangle$, where R_i is an m_i-ary relation symbol. Let $k \geq 1$ be a constant and suppose that we have formulas $U(x_1 \dots x_k)$, $\varphi_1(x_1 \dots x_{k \cdot m_1}) \dots \varphi_p(x_1 \dots x_{k \cdot m_p})$, from $L[\tau_1]$ with the indicated free variables. Thus U picks out certain k-tuples of points and φ_i is an m_i-ary relation on these k-tuples.

These formulas induce a map f from structures of type τ_1 to structures of type τ_2. The definition for f is:

$$\text{Universe}[f(S)] = \{ \langle t_1 \dots t_k \rangle \in S^k \mid S \models U(t_1 \dots t_k) \}$$
$$R_i^{f(S)} = \{ (u_1 \dots u_{m_i}) \mid S \models \varphi_i(u_1 \dots u_{m_i}) \}$$

In words, the universe of f(S) consists of those k-tuples from the universe of S which satisfy U. The relation R_i is interpreted in f(S) as those m_i-tuples of k-tuples from S which when concatenated satisfy the $k_i \cdot m_i$-ary formula φ_i.

We define a k-ary interpretation between structures to be such a map f: $Mod(\tau_1) \to Mod(\tau_2)$, induced by formulas $U, \varphi_1 - \varphi_p$. Recall that $Mod(\tau)$ is the class of all structures of type τ.

Sometimes we want to consider only those structures of type τ which fit a certain format. For example we might consider those graphs $\langle V, E, \leq \rangle$ such that $\leq$ is a linear ordering of V. We can express this property with a sentence I and let $Mod(\tau, I)$ be the class of structures of type τ which satisfy I.

If f: $Mod(\tau_1) \to Mod(\tau_2)$ is an interpretation between structures and satisfies the condition:

$$S \models I_1 \quad \Rightarrow \quad f(S) \models I_2$$

then we call f an interpretation between theories (IBT) I_1 and I_2.

Example: Let $\tau_1 = \langle F(\cdot,\cdot), \leq, F(\cdot), L(\cdot) \rangle$ and $\tau_2 = \langle E(\cdot,\cdot), A(\cdot), B(\cdot) \rangle$. We are thinking of τ_1 as ordered graphs with first point satisfying F and last point satisfying L. We want τ_2 to be the type of graphs having unique points satisfying A(·), and B(·). Informally define I_1 and I_2 as follows:

I_1 ≡ "$\leq$ is a total ordering and F(x) holds iff x is first and L(x) holds iff x is last under $\leq$."

I_2 ≡ "There exist unique points a and b with A(a) and B(b)."

Define an interpretation between theories, f: $Mod(\tau_1, I_1) \to Mod(\tau_2, I_2)$, as follows:

$$U(x_1, x_2, x_3) \equiv (x_1 = x_1)$$

$$\varphi_1(x_1, x_2, x_3, y_1, y_2, y_3) \equiv [x_1 = y_1 \ \& \ x_2 = y_2 \ \& \ E(x_3, y_3)] \text{ or } [x_2 = x_3 \ \& \ y_1 = y_3 \ \& \ Suc_2(x_1, x_2, y_1, y_2)]$$

$$\varphi_2(x_1, x_2, x_3) \equiv F(x_1) \ \& \ F(x_2) \ \& \ F(x_3)$$

$$\varphi_3(x_1, x_2, x_3) \equiv L(x_1) \ \& \ L(x_2) \ \& \ L(x_3)$$

U(x) always holds so $Universe[f(S)] = (Universe[S])^3$. In the definition of φ_1, $Suc_2(x_1, x_2, y_1, y_2)$ means that $\langle y_1, y_2 \rangle$ is the immediate successor of $\langle x_1, x_2 \rangle$ in the lexicographical ordering. In symbols:

$$Suc_2(x_1, x_2, y_1, y_2) \equiv [x_1 = y_1 \ \& \ Suc(x_2, y_2)] \text{ or } [L(x_2) \ \& \ F(y_2) \ \& \ Suc(x_1, y_1)]$$

$$Suc(x,y) \equiv x \leq y \ \& \ x \neq y \ \& \ \forall z [x \leq z \Rightarrow y \leq z \text{ or } x = z]$$

The idea is that f(S) consists of n^2 copies of S, one for each pair of vertices $\langle u,v \rangle$. Each copy may be entered at $\langle u,v,u \rangle$ and exited from $\langle u,v,v \rangle$. Thus there is a path from a to b in f(S) if and only if there is a path between each pair of vertices from S. That is for each S in $Mod(\tau_1, I_1)$, f(S) is in $Mod(\tau_2, I_2)$, and S is connected iff f(S) has the GAP property. Thus we have used an interpretation between theories to reduce connectivity to GAP.

Section 5.2: Var &Sz Reductions.

Let $f: Mod(\tau_1) \to Mod(\tau_2)$ be an Interpretation Between Structures. Notice that f induces a map $F: L[\tau_2] \to L[\tau_1]$. F replaces each variable x by the k variables $x_1 \dots x_k$, and each relaton symbol R_i by the formula φ_i. In symbols:

$$\begin{aligned} F[R_i] &= \varphi_i \\ F[\exists x M(x)] &= \exists x_1 \dots x_k \; U(x_1 \dots x_k) \;\&\; F[M](x_1 \dots x_k) \\ F[\forall x M(x)] &= \forall x_1 \dots x_k \; U(x_1 \dots x_k) \Rightarrow F[M](x_1 \dots x_k) \end{aligned}$$

It is easy to see that if S is in $Mod[\tau_1]$, then

$$S \models F[M] \quad \leftrightarrow \quad f(S) \models M \; .$$

Suppose that IIFT f is determined by $U, \varphi_1 \dots \varphi_p$. Then the size of f is the sum of the sizes of $U, \varphi_1 \dots \varphi_p$. Also the number of variables of f is the maximum number of variables in any of $U, \varphi_1 \dots \varphi_p$.

Definition: Let A and B be subsets of $Mod[\tau_1]$ and $Mod[\tau_2]$ respectively. We say that A is Var &Sz[v(n),z(n)] reducible to B (in symbols, $A \leq_{Var\ \&Sz[v(n),z(n)]} B$) if there exists a constant k and a uniform sequence, $f_1, f_2 \dots$, of k-ary IIFT's from $Mod[\tau_1]$ to $Mod[\tau_2]$ such that:

(a): $Size[f_n] \leq z(n)$ and $Var[f_n] \leq v(n)$ $\qquad n = 1,2, \dots$

(b): $\forall S \in Mod[\tau_1] \big(|S| \leq n \Rightarrow [S \in A \leftrightarrow f(s) \in B] \big)$

Thus our reductions consist of a uniform sequence of interpretations of structures of type τ_1 as structures of type τ_2. These reductions honor Var &Sz in the following sense:

Theorem 5.1: Let A and B be problems with B in Var &Sz[v(n),z(n)]. Suppose $A \leq_{Var\ \&Sz[u(n),s(n)]} B$. Then A is in Var &Sz[k•u(n^k)+u(n), z(n^k)+s(n)] where k is the arity of the reductions.

proof: Let $P_1, P_2 \ldots$ be the Var &Sz[v(n),z(n)] sentences that express property B. Let $S \in Mod[\tau_1]$, |S| = n. Then using the n^{th} IBT, f_n we have:

$$\begin{aligned} S \in A &\leftrightarrow f_n(S) \in B \\ &\leftrightarrow f_n(S) \models P_{n^k} \\ &\leftrightarrow S \models F_n[P_{n^k}] \end{aligned}$$

Thus $\{ F_n[P_{n^k}] \mid n \geq 1 \}$ is a uniform sequence of sentences expressing property A. Computing Size and number of variables needed we get:

$$\begin{aligned} Size[F_n[P_{n^k}]] &= O[z(n^k)+s(n)] \\ Var[F_n[P_{n^k}]] &= k \bullet v(n^k)+u(n) \end{aligned}$$

Note that in the computation of the size we use the abbreviation trick to only write out each φ_i once. Otherwise the s(n) would be a multiplicative factor. ∎

Here are two cases of Theorem 5.1 with values substituted in for u, s, v, and z:

Corollary 5.1a:

1. If $A \leq_{Var[k]} B$ and $B \in Var[k]$, then $A \in Var[k]$.
2. If $A \leq_{Size[\log n]} B$ and $B \in Size[\log^i(n)]$, then $A \in Size[\log^i(n)]$.

In Theorem 1.2 we essentially gave a Size[logn] reduction of an arbitrary problem in NSPACE[logn] to GAP. Recall the construction. Let A be any problem accepted by Turing machine M in NSPACE[logn]. Suppose that problems of size n require at most c•log(n) bits for the work tape. Then an ID of M can be coded with k = 2c+3 vertices, $\langle t_1 \ldots t_c, h_1 \ldots h_c, r_1, r_2, q \rangle$, where $t_1 \ldots t_c$ record the contents of the work tape, $h_1 \ldots h_c$ record the head position, r_1 and r_2 give the input head position and q gives the state.

The edge relation, $\varphi_1(ID_1, ID_2)$ is a formula saying that ID_2 follows from ID_1 in one move of M. We showed in Theorem 1.2 and Lemma 1.3 that φ_1 may be written in Size[logn]. The predicate U

saying that the k-tuple is of the correct form, and the monadic predicates A and B picking out the initial and final configurations can all be expressed in Size[logn]. We have shown:

Proposition 5.2: GAP is complete for NSPACE[logn] via Size[logn] reductions.

Of course this proposition is somewhat silly because NSPACE[logn] is contained in Size[logn], i.e. the reductions are already strong enough to solve the problem A. In some sense the log n quantifiers for the reduction aren't needed at all. If we had the predicate ON[x,y] meaning that bit y of vertex x is On, in addition to the predicate Suc to indicate the ordering of vertices, then a single IBT would translate all instances of A to instances of GAP.

Note also that Proposition 5.2 as well as the above remark go through for reducing PTIME to AGAP. The only difference is that another predicate Or(-) must be added to indicate the "or" nodes. Or(ID) will be true just if the associated state is an existential state of M.

The idea of translating one problem into another seems a natural notion for reductions. This chapter only touches the suface of the subject. We hope that someone may be interested to study the relations between IBT's and other reductions, and to study some of the implications of the existence or non-existence of certain IBT's.

Chapter 6

Towards Lower Bounds with Successor

In chapters 3 and 4 we have demonstrated some lower bounds on expressibility without successor (cf. Theorems 3.5,4.5.4.6,4.7 .) Our tools were Ehrenfeucht-Fraisse games to prove quantifier rank lower bounds, and alternating pebbling games to also prove lower bounds on number of variables needed.

In this chapter we discuss attempts to extend these results in include successor. Section 6.1 give some negative results and shows in particular that for ordered graphs quantifier rank lower bounds will no longer be of an: help

Section 6.2 suggests another modification of Ehrenfeucht-Fraisse games which we call the Separability game. We give evidence in Section 2 that these games may be a reasonable combinatorial tool to prove the desired lower bounds with successor. Section 6.3 then goes on to give examples and discussion of the separability game. Finally in Section 6.4 we list some other ideas for proving things about successor.

Section 6.1: Quantifier Rank Doesn't Do It.

By an ordered graph we mean a graph which comes with a valid successor relation. The following proposition shows that any property whatsoever of ordered graphs can be expressed in quantifier rank $\log(n)+3$.

Proposition 6.1: Let C be any set of ordered graphs. Then for all n there exist sentences S_n of quantifier rank $\log(n)+3$ and with three distinct variables such that for all ordered graphs G of size $\leq n$,

$$G \in C \quad \leftrightarrow \quad G \models S_n .$$

proof: First we show that for any $i_0 \leq n$, we can write the formula $N_{i_0}(x)$, which means, "x is vertex number i_0 in the Suc ordering," in quantifier rank $\log(n)+1$ and with three distinct variables. This is done by inductively defining the formulas $P_i(x,y)$ to mean that there is a successor path of length exactly i from x to y.

$$\begin{aligned} P_1(x,y) &= Suc(x,y) \\ P_{2n-1}(x,y) &= \exists z[\, P_{n-1}(x,z) \,\&\, P_n(z,y)\,] \\ P_{2n}(x,y) &= \exists z[\, P_n(x,z) \,\&\, P_n(z,y)\,] \end{aligned}$$

Now we identify the i^{th} point by saying that there is a path of length i from the first point to it:

$$N_i(x) = \exists z[\, P_{i-1}(z,x) \,\&\, \forall y(\neg Suc(y,z))\,]$$

$N_n(x)$ has quantifier rank $\log(n)+1$ and could also be written with $O[\log n]$ quantifiers using the abbreviation trick.

Now using $N_i(x)$ we can completely describe any graph G as follows :

$$F_G = \bigwedge_{i,j=1\ldots n} \exists x\, \exists y\, [\, N_i(x) \,\&\, N_j(y) \,\&\, F^{i,j}(x,y)\,]$$

Here $F^{i,j}(x,y)$ is $F(x,y)$, or $\neg F(x,y)$ according as $F(v_i,v_j)$ holds or does not hold in G. Note that F_G has quantifier rank $\log(n)+3$. Let $C_n = \{\, G \mid G \in C \,\&\, |G| \leq n \,\}$. We define S_n as the disjunction over all G in C_n of F_G, i.e.

$$S_n = \bigvee_{G \in C_n} F_G$$

This is the desired complete description of C_n. Although it may have length 2^{n^2}, S_n has quantifier rank only $\log(n)+3$. ∎

Section 6.2: Separability.

In spite of the above proposition there is still hope. Recall that from Theorem 3.5 we have a pair of structures G_n' and H_n' which are $2^{(\log n)^{1/2}}$ equivalent but differ on the AGAP property. We conjecture that AGAP is not in Size[$2^{(\log n)^{1/2}}$].

Consider the set of all posssible orderings of a graph G:

$$S(G) \quad = \quad \{ \langle G, Suc_i \rangle \mid Suc_i \text{ is a successor relation on } G. \}$$

Thus $S(G_n')$ and $S(H_n')$ are families of ordered structures which we suspect cannot be separated by a sentence with $2^{(\log n)^{1/2}}$ quantifiers. To make the notion "separated" precise we give the following

Definition: Let M and N be families of structures of the same finite type, τ. We say that M and N are k-inseparable if there is no sentence, F, from $L[\tau]$ with k quantifiers such that:

$$M \models F \qquad \text{and} \qquad N \models \neg F ,$$

i.e. every structure in M satisfies F and no structure in N does. Otherwise M and N are k-separable.

Clearly if we could show that $S(G_n')$ and $S(H_n')$ are $\log^k(n)$ inseparable it would follow that AGAP is not in Size[$\log^k(n)$]. The notion, "$AGAP_n$ and $\neg AGAP_n$ are O[f(n)]-separable," would be the same as the condition, "AGAP is in Size[f(n)]," if we had omitted the uniformity requirement in the definition of Size. Thus the following generalization of Theorem 2 holds:

Proposition 6.2: Let C be any set of ordered graphs. Then:

a. Suppose C is in $NSPACE^T[\log n]$ for some sparse oracle set T. Then C_n is O[log n]-separable from $\neg C_n$, for every n.

b. Suppose C_n is O[log n]-separable from $\neg C_n$, for every n. Then there is a sparse oracle, T, such that C is in $DSPACE^T[\log^2(n)]$.

proof: T is a sparse set if there are at most n^k objects in T of length n. A $SPACE^T[f(n)]$ machine has a size f(n) query tape on which it may write words and ask if they are in T. The proof of this proposition

is similar to that of Theorem 2. The differences are:

For part (a), we must code into F_n the n^k elements of T that the logn space Turing machine can look at. Thus the formula, $P_i(ID_a, ID_b)$, saying that ID_b follows from ID_a in one step of M^T must include the disjunction over n^k possible questions to the oracle. However all quantifiers may be placed outside this disjunction so the number of quantifiers is unchanged, and this is what separability measures.

For part (b), let F_n be the log(n) quantifier sentence which separates C_n from $\neg C_n$. We must code F_n into $T \cap \{ w \mid |w| = 2^{\log^2(n)} \}$. Note that any sentence with f(n) quantifiers and binary predicates is equivalent to some sentence of length $2^{f^2(n)}$. ∎

Proposition 6.2 is encouraging because it suggests that PTIME complete properties may be O[logn]-inseparable from their complements. We close this section with a modified version of Ehrenfeucht-Fraisse games which test for separability:

Definition: Given families of structures, M and N, of the same finite type, we define the k-move separability game on M and N as follows:

On each of the k moves Player I chooses a point from each structure on one side or the other. Player II then chooses a corresponding point from each structure on the other side. II is allowed to make copies of structures so that she may choose several different answers from the same structure.

We say that Player II wins if there is a pair of structures and sequences of moves, $\langle G_i, m_1^i \ldots m_k^i \rangle$ and $\langle H_j, n_1^j \ldots n_k^j \rangle$ one from each side such that the map which sends constants from G_i to constants from H_j and maps m_r^i to n_r^j is an isomorphism of the induced substructures.

Theorem 6.3: Player II has a winning strategy for the k move game on M and N iff M and N are k-inseparable.

proof: By induction on k.

k=0. Here if Player II wins then there is a pair of structures $G \in M$ and $H \in N$ whose constants are isomorphic. It follows that G and H satisfy all the same quantifier free formulas and so M and N are

0-inseparable. Conversely if there is no such pair then the quantifier free formula, F_0, which is a disjunction of all the isomorphism types of constants from M is satisfied by all of M and none of N.

Inductively, assume that the $r+1$ quantifier formula $\exists x P(x)$ is true in M and false in N. Then Player I's first move will be to choose a point m_1^i from each $G_i \in M$ in such a way that $G_i \models P(m_1^i)$. No matter what II does, no structure $H_j \in N$ will satisfy $P(n_1^j)$. Think of the language as now having a new constant symbol c_1. Thus $\langle M, m_1 \rangle \models P(c_1)$ and $\langle N, n_1 \rangle \models \neg P(c_1)$ so by induction Player I wins.

Conversely assume that M and N are $r+1$-inseparable and let Player I choose m_1^i from each $G_i \in M$. Let $F_1 \ldots F_s$ be a list of all the r quantifier formulas with the new symbol c_1 that are true for each structure in M, that is:

$$\langle M, m_1 \rangle \models F_i(c_1) \qquad i = 1 \ldots s \ .$$

Therefore, $$M \models \exists x F_i(x) \qquad i = 1 \ldots s \ .$$

Since $\exists F_i(x)$ cannot separate M from N there must be some H_i in N such $H_i \models \exists x F_i(x)$. Thus Player II can play these s witnesses from the appropriate H_i's and forget about the rest of N. Note that this is where the making of copies is needed in case $H_i = H_j$ for some $i \neq j$. Thus Player II can preserve the condition that M and N are r-inseparable and so by induction she will win. ∎

Section 6.3: Playing the Separability Game.

As a first example of the separability game consider structures $S = \langle \{1 \ldots n\}, M, \leq \rangle$ consisting of an n point universe $\{1 \ldots n\}$ with its usual ordering, $\leq$, and a monadic relation M. S can be thought of as a binary string of length n with a 1 in the i^{th} place whenever M(i) holds.

Let $\quad Even_n = \{S = \langle \{1 \ldots n\}, M, \leq \rangle \mid |M| \text{ is even}\}$

and $\quad Odd_n = \{S \mid |M| \text{ is odd}\}$

be the sets of these structures inwhich M holds on an even (resp. odd) number of points. Let's play the separability game on ODD_n and $Even_n$. The game proceeds as follows:

Figure 6.1: A Separability Game

Even	Odd
$G_1, n/2, \ldots$	$H_{1,1}, 1, \ldots$
	...
	$H_{1,n}, n, \ldots$
...	...
$G_k, n/2, \ldots$	$H_{k,n}, n, \ldots$

Player I chooses point n/2 in each structure on the left side. Player II's optimal strategy is always to choose each possible point in every structure on the other side.

Notice that after the first move each structure, $\langle G_i, c_1 \rangle$ on the left has the property that either $[1,c_1]$ and $[c_1+1,n]$ are both even (i.e. have an even number of points satisfying M) or they are both odd.

Player I can mark each structure on the left which is in the first category. For example, Player I can make the first choice equal to the second choice for all the structures he wishes to mark, and not for the others. After Player II answers, the only possible isomorphisms willl be between marked structures (those for which $c_1 = c_2$) and between unmarked structures.

Thus on the second move Player I marks those structures on the left so that $[1,c_1]$ and $[c_1+1,n]$ are both even. Player II answers by making a copy of each structure on the right and marking one copy of each pair.

Note that on the right at least one of $[1,c_1]$ and $[c_1+1,n]$ is wrong. That is the structure is marked and one of the intervals is odd, or the structure is not marked and one of the intervals is even.

Player I now marks all the structures on the right for which $[1,c_1]$ has the wrong parity. After II responds the game has been broken into four parts such that for each part all the chosen intervals on the left are of length $\leq n/2$ and of a fixed parity. All the matching intervals on the right are of a different parity.

Player I has succeeded in breaking the problem in half using 3 moves. Thus 3log(n) moves suffice

for Player I to win the game: when the differing intervals are of length 1 on the left there can be no isomorphisms between the left and the right.

The above sample game exposes certain differences between the separability game and the usual Ehrenfeucht-Fraisse game. For one thing the idea of marking is crucial in the separability game. What Player I accomplishes in the above 3log(n) moves is to subdivide the game $Even_n$ versus Odd_n into a large number of subpieces so that each pair at the bottom, P,Q , has the property that for some selected point i, M(i) holds for all of P and none of Q or vice-versa.

Notice that the separability game emphasizes the need to alternate quantifiers when their number is restricted. Recall that the alternation of quantifiers coresponds to Player I switching sides of the board for his moves. Clearly if Player I can only play on one side then n moves are required because any configuration of n-1 points occuring in some structure in Even occurs also in a structure from Odd.

At this point we have only scattered bits of information about the separability game. We leave it here as a jumping off point for further research. We urge others to study it, hoping that the separability game may become a viable tool for ascertaining lower bounds.

Section 6.4: Other Ways to Approach Successor.

We feel that proving lower bounds on expressibility with successor is very difficult but also very important. Here is a list of several possible approaches to this problem:

1. Separability Game: Already discussed in this chapter.

2. Average Successor: Graphs G_n and H_n of Theorem 3.5 can be distinguished by a special ordering. For example, if the given ordering happened to place all points from which d is reachable in front of the points from which it is not, then this fact could be asserted by a fixed first order sentence. Thus "smart" successors can check polynomial time complete problems. We suspect, however, that the short sentences true for an average ordering of G_n are the same as for an average ordering of H_n. Perhaps counting techiniques such as those in Section 4.3

could help prove such a conjecture.

3. Generic Successor: A similar idea to (2) would be to get at the average successor from a syntactic point of view. Perhaps one could modify the notion of forcing in model theory (an adaption by Abraham Robinson of work of Paul Cohen) to determine which short sentences are true for a "generic" successor.

4. Approximation of Successor from Below: The crucial bad property of Successor is that it enables us to pick out a point by number and thus distinguish any two objects with logn quantifiers. However, when we throw it away we also lose certain desirable things such as the ability to count a bunch of indistinguishable points, and the ability to add up the parities of a bunch of switches. These last two things are tasks which a DSPACE[logn] Turing machine can of course perform. Throwing them away makes our lower bounds suspicious. Perhaps we can add to our language such things as the ability to count, e.g. we could add the quantifiers:

$(\exists$ m x's$)A(x)$, meaning, "There exist at least m distinct points, x, such that A(x)."

A study of upper and lower bounds for expressibility in languages between $L[\tau]$ and $L[\tau \cup \{Suc\}]$ would be interesting, and a step towards achciving the desired bounds on Var &Sz with successor.

Chapter 7

Conclusions and Directions for Furture Research

We have shown that first order expresibility is an alternate view of computational complexity. In particular, the number of symbols and of distinct variables needed to express a property, C, is closely tied to the space and to the logarithm of the time needed to check if C holds for a given input. This realization leads to several exciting possibilities:

First there is hope for proving that particular properties are not expressible without a certain number of symbols or variables. In Chapters 3 and 4 we proved some lower bounds on expressibility without successor. It would be interesting and worthwhile to learn just which graph properties are easily expressible in the language without successor.

As we saw in Chapter 6, when successor is added the problem of proving lower bounds becomes much more dificult. Some possible methods of attack were mentioned there. We feel that even if such approaches as the separability game never lead to the desired lower bounds they will still provide new insight into expressibility and the power of alternation.

The fact that complexity is closely tied to expressibility suggests an approach to algorithm design: all one has to do is to efficiently express the condition to be checked for. Much work towards this goal has already been done, a good example is the SETL programming language. (See [KeSc75].) This dissertation provides some theoretical justification for such an approach. An efficiently expressed condition leads to an efficient algorithm. Furthermore we have discussed combinatorial games which give lower bounds on the efficiency of expressions. They may be used to help design optimal descriptions of the tasks to be performed. First order descriptions are most closely tied to parallel algorithms, for example, "A & B" is an instruction to split into two processors, one checking A, the

other Π. We anticipate research connecting expressibility and parallel algorithms, hopefully leading to efficient algorithms, and/or lower bounds on time, space, and number of processors needed.

In this thesis we have compared the complexity of computations with the ease of expressibility in the language of mathematics. We have observed that the classic complexity questions may be viewed as natural problems in mathematics unaffected by the quirks of a particular model of computation such as the Turing machine. We hope that such work may lead to more contact and cross fertilization of ideas between Theoretical Computer Science and the rest of mathematics.

We close with a list of some of the open problems we would like to see solved:

1. Improved Bounds Relating Expressibility to Turing Machine Complexity (cf. Chapters 1, 2).
   a. How many variables are needed to simulate DTIME[n^k]? We suspect that corollarty 2.11 could be improved to something like DTIME[n^k] $\subseteq$ Var[$k+3$].
   b. Improve the simulations of section 2.4. Once they are improved try to prove optimality.
   c. Study corollary 2.8 trying to amass some evidence concerning which of the containments are proper.

2. Lower Bounds Without Successor (cf. Chapters 3, 4).
   a. In theorem 3.2 we showed the NSPACE[log n] $\subseteq$ Size[log n] is a tight bound. Try to prove similar results for NSPACE[$f(n)$] $\subseteq$ Size[$f(n)^2/\log(n)$] $\subseteq$ DSPACE[$f(n)^2$].
   b. Theorem 3.5 gives a lower bound for AGAP of Size(w.o. Suc)[$2^{(\log n)^{1/2}}$], whereas the best upper bound we know of is Size[n]. Try to lessen this descrepency.
   c. Find upper and lower bounds on Size(w.o. Suc) for other problems besides AGAP. Possibilities abound: planarity, graph homeomorphism, vertex matching, three colorability, etc.
   d. As in (c) find upper and lower bounds for Variables(w.o. Suc).

3. Reductions and Complete Sets (cf. Chapter 5).

   a. Relate the power of Interpretations Between Theories to the power of other reductions between problems.

   b. The universal games studied in [ChSt76] and [Rei79] seem to provide natural complete problems for certain expressibility classes. So do instances of the Ehrenfeucht-Fraisse games. Make the above two statements more precise and find other complete sets as well.

   c. The private alternation and multiple alternation studied in [Rei79] and [PeRe79] suggest the analagous notion of "private quantifiers." Does this approach add insight into Reif and Peterson's new alternation classes?

4. Dealing With Successor (cf. Chapter 6).

   a. Learn to play the Separability game, well.

   b. Learn about which sentences are true for an average or generic successor.

   c. Study expressibility with features weaker than Suc, e.g. the ability to count.

   d. Study the notion of adding an arbitrary relation with a certain property, as for example adding a Successor relation. In particular we can add a deterministically defined relation called a "marking" which allows us to express PTIME properties such as AGAP. What else can be said? Do relations weaker than Suc still allow us to simulate Turing machines? Can others give us more than PTIME?

5. Other Things.

   a. Relate Parallel Algorithms to expressibility. Find optimal expressions of graph properties. What measure of expressibility (if any) corresponds to number of processsors?

   b. Pursue the notion of designing efficient expressions as an alternative to designing efficient algorithms.

Bibliography

[AHU74]: Aho,A., Hopcroft,J., Ullman,J., The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

[BlHa79]: Blass,A., Harary,F., "Properties of Almost All Graphs and Complexes," J. of Graph Theory Vol. 3, 1979, pp. 225-240.

[Bor77]: Borodin,A., "On Relating Time and Space to Size and Depth," SIAM J. on Computing, Vol. 6, No. 4, Dec. 1977, pp. 733-744.

[ChSt76]: Chandra,S., Stockmeyer,L., "Alternation," Proc. 17th FOCS, 1976, pp. 98-108.

[Ehr61]: Ehrenfeucht,A., "An Application of Games to the Completeness Problem for Formalized Theories," Fund. Math, Vol. 49, 1961, pp. 129-141.

[End72]: Enderton,H., A Mathematical Introduction to Logic, Academic Press, 1972.

[Fag74]: Fagin,R., "Generalized First-Order Spectra and Polynomial-Time Recognizable Sets," in Complexity of Computation, (ed. R.Karp), SIAM-AMS Proc. 7, 1974, pp. 43-73.

[Fag76]: _____, "Probabilities on Finite Models," JSL Vol 41, No. 1, 1976, pp. 50-58.

[FiRa74]: Fischer,M., Rabin,M., "Super-Exponential Complexity of Presburger Arithmetic," in Complexity of Computation, (ed. R.Karp), SIAM-AMS Proc. 7, 1974, pp. 27-41.

[Fra54]: Fraisse,R., "Sur les Classifications des Systems de Relations," Publications Sc. de l'Universite d'Alger, 1, 1954.

[Gol77]: Goldschlager,L., The Monotone and Planar Circuit Value Problems are Log Space Complete for P, SIGACT News, Vol. 9, No. 2, 1977.

[HIM78]: Hartmanis,J., Immerman,N., Mahaney,S., "One-Way Log Tape Reductions," Proc. 19th FOCS, 1978, pp. 65-72.

[HPV77]: Hopcroft,J., Paul,W., Valiant,L., "On Time Versus Space," JACM, Vol. 24, No. 2, 1977, pp. 332-337.

[Imm79]: Immerman,N., "Length of Predicate Calsulus Formulas as a New Complexity Measure," Proc. 20th FOCS, 1979, pp. 337-347.

[Imm80]: _____, "Number of Quantifiers is Better than Number of Tape Cells," to appear in JCSS, 1980.

[Jon75]: Jones,N., "Space-Bounded Reducibility Among Combinatorial Problems," JCS 11, 1975, pp. 68-75.

[KeSc75]: Kennedy,K., Schwartz,J., "An Introduction to the Set Theoretical Language SETL," Comp. & Maths with Appls, Vol. 1, pp. 97-119, Pergamon Press, 1975.

[Koz76]: Kozen,D., "On Parallelism in Turing Machines," Proc. 17th FOCS, 1976, pp. 89-97.

[PeRe79]: Peterson,G., Reif,J., "Multiple-Person Alternation," Proc. 20th FOCS, 1979, pp. 307-311.

[Pip79]: Pippenger,N., "On Simultaneous Resource Bounds," Proc. 20th FOCS, 1979, pp. 307-311.

[Rei79]: Reif,J., "Universal Games of Incomplete Information," Proc. 11th SIGACT, 1979, pp. 288-308.

[Ru79a]: Ruzzo,W, "Tree-Size Bounded Alternation," Proc. 11th SIGACT, 1979, pp. 352-359.

[Ru79b]: _____, "On Uniform Circuit Complexity," Proc. 20th FOCS, 1979, pp. 312-318.

[Sav70]: Savitch,W., "Maze Recognizing Automata and Nondeterministic Tape Complexity," JCSS 7, 1973, pp. 389-403.

[SaSt79]: Savitch, W., Stimson, M., "Time Bounded Random Access Machines with Parallel Processing," JACM Vol. 26, No. 1, 1979, pp. 103-118.

[Sto77]: Stockmeyer, L., "The Polynomial-Time Hierarchy," Theoretical Comp. Sci. 3, 1977, pp. 1-22.

[Sud78]: Sudborough, I., "On the Tape Complexity of Deterministic CFL's," JACM Vol. 25, No. 3, 1978, pp. 405-414.